

# **Coherent PDF .NET API and Command Line Tools**

**User Manual**  
Version 2.5 (January 2022)



**Coherent Graphics Ltd**

For bug reports, feature requests and comments, email  
contact@coherentgraphics.co.uk

©2022 Coherent Graphics Limited. All rights reserved. ISBN 978-0957671140

Adobe, Acrobat, and Adobe PDF are registered trademarks of Adobe Systems Incorporated.  
Windows, Powerpoint and Excel are registered trademarks of Microsoft Corporation.

# Quickstart Examples

Just a few of the facilities provided by the Coherent PDF Command Line Tools. See each chapter for more commands and full details.

## Chapter 1: Basic Usage

```
cpdf in.pdf 1-3,6 -o out.pdf
```

Read `in.pdf`, select pages 1, 2, 3 and 6, and write those pages to `out.pdf`.

```
cpdf in.pdf even -o out.pdf
```

Select the even pages (2, 4, 6...) from `in.pdf` and write those pages to `out.pdf`.

```
cpdf -merge in.pdf in2.pdf AND -add-text "Copyright 2021"  
-o out.pdf
```

Using `AND` to perform several operations in order, here merging two files together and adding a copyright stamp to every page.

```
cpdf -args control.txt
```

Read `control.txt` and use its contents as the command line arguments for `cpdf`.

## Chapter 2: Merging and Splitting

```
cpdf -merge in.pdf in2.pdf -o out.pdf
```

Merge `in.pdf` and `in2.pdf` into one document, writing to `out.pdf`.

```
cpdf -split in.pdf -o Chunk%%.pdf -chunk 10
```

Split `in.pdf` into ten-page chunks, writing them to `Chunk001.pdf`, `Chunk002.pdf` etc.

```
cpdf -split-bookmarks 0 in.pdf -o @N.pdf
```

Split `in.pdf` on bookmark boundaries, writing each to a file whose name is the bookmark label.

## Chapter 3: Pages

```
cpdf -scale-page "2 2" in.pdf -o out.pdf
```

Scale both the dimensions and contents of `in.pdf` by a factor of two in x and y directions.

```
cpdf -scale-to-fit usletterportrait in.pdf -o out.pdf
```

Scale the pages in `in.pdf` to fit the US Letter page size, writing to `out.pdf`

```
cpdf -shift "26pt 18mm" in.pdf -o out.pdf
```

Shift the contents of the page by 26 pts in the x direction, and 18 millimetres in the y direction, writing to `out.pdf`

```
cpdf -rotate-contents 90 in.pdf -o out.pdf
```

Rotate the contents of the pages in `in.pdf` by ninety degrees and write to `out.pdf`.

```
cpdf -crop "0 0 600pt 400pt" in.pdf -o out.pdf
```

Crop the pages in `in.pdf` to a 600 pts by 400 pts rectangle.

## Chapter 4: Encryption and Decryption

```
cpdf -encrypt 128bit fred joe in.pdf -o out.pdf
```

Encrypt `in.pdf` using 128bit PDF encryption using the owner password `fred` and the user password `joe` and writing the encrypted file to `out.pdf`

```
cpdf -decrypt in.pdf owner=fred -o out.pdf
```

Decrypt `in.pdf` using the owner password, writing to `out.pdf`.

## Chapter 5: Compression

```
cpdf -compress in.pdf -o out.pdf
```

Compress the data streams in `in.pdf`, writing the result to `out.pdf`.

```
cpdf -decompress in.pdf -o out.pdf
```

Decompress the data streams in `in.pdf`, writing to `out.pdf`.

```
cpdf -squeeze in.pdf -o out.pdf
```

Squeeze `in.pdf`, writing to `out.pdf`. Squeezing rearranges the structure of the PDF file to save space.

## Chapter 6: Bookmarks

```
cpdf -list-bookmarks in.pdf
```

List the bookmarks in `in.pdf`.

```
cpdf -add-bookmarks bookmarks.txt in.pdf -o out.pdf
```

Add bookmarks in the same form from a prepared file `bookmarks.txt` to `in.pdf`, writing to `out.pdf`.

## Chapter 7: Presentations

```
cpdf -presentation in.pdf 2-end -trans Split -duration 10
-o out.pdf
```

Use the Split style to build a presentation from the PDF `in.pdf`, each slide staying 10 seconds on screen unless manually advanced. The first page, being a title does not move on automatically, and has no transition effect.

## Chapter 8: Logos, Watermarks and Stamps

```
cpdf -stamp-on watermark.pdf in.pdf -o out.pdf
```

Stamp the file `watermark.pdf` on to each page of `in.pdf`, writing the result to `out.pdf`.

```
cpdf -topleft 10 -font Courier
-add-text "Page %Page\nDate %d-%m-%Y" in.pdf -o out.pdf
```

Add a page number and date to all the pages in `in.pdf` using the Courier font, writing to `out.pdf`.

## Chapter 9: Multipage Facilities

```
cpdf -twoup-stack in.pdf -o out.pdf
```

Two up impose the file `in.pdf`, writing to `out.pdf`.

```
cpdf -pad-after in.pdf 1,3,4 -o out.pdf
```

Add extra blank pages after pages one, three and four of a document.

## Chapter 10: Annotations

```
cpdf -list-annotations in.pdf
```

List the annotations in a file `in.pdf` to standard output.

```
cpdf -copy-annotations from.pdf in.pdf -o out.pdf
```

Copy the annotations from `from.pdf` to `in.pdf`, writing to `out.pdf`.

## Chapter 11: Document Information and Metadata

```
cpdf -set-title "The New Title" in.pdf -o out.pdf
```

Set the document title of `in.pdf`, writing to `out.pdf`.

```
cpdf -hide-toolbar true in.pdf -o out.pdf
```

Set the document `in.pdf` to open with the PDF Viewer's toolbar hidden, writing to `out.pdf`.

```
cpdf -set-metadata metadata.xml in.pdf -o out.pdf
```

Set the metadata in a PDF `in.pdf` to the contents of the file `metadata.xml`, and write the output to `out.pdf`.

```
cpdf -set-page-layout TwoColumnRight in.pdf -o out.pdf
```

Set the document `in.pdf` to open in PDF Viewer showing two columns of pages, starting on the right, putting the result in `out.pdf`.

```
cpdf -set-page-mode FullScreen in.pdf -o out.pdf
```

Set the document `in.pdf` to open in PDF Viewer in full screen mode, putting the result in `out.pdf`.

## Chapter 12: File Attachments

```
cpdf -attach-file sheet.xls in.pdf -o out.pdf
```

Attach the file `sheet.xls` to `in.pdf`, writing to `out.pdf`.

```
cpdf -remove-files in.pdf -o out.pdf
```

Remove any attachments from `in.pdf`, writing to `out.pdf`.

## Chapter 13: Working with Images

```
cpdf -image-resolution 600 in.pdf
```

Identify and list any image used at less than 600dpi.

```
cpdf -extract-images in.pdf -im /usr/bin/magick -o output/%%%
```

Extract images from `in.pdf` to directory `output` (with the help of `imagemagick`).

## Chapter 14: Fonts

```
cpdf -list-fonts in.pdf
```

List the fonts in use, and what pages they are used on.

```
cpdf -gs /usr/bin/gs -embed-missing-fonts in.pdf -o out.pdf
```

Embed missing fonts (with the help of Ghostscript).

## Chapter 15: PDF and JSON

```
cpdf in.pdf -output-json -output-json-parse-content-streams  
-o out.json
```

Write the PDF in JSON format to the given file, parsing its content streams into individual JSON objects too.

```
cpdf -j in.json -o out.pdf
```

Load a PDF in JSON format, writing to an output PDF.



## Chapter 16: Optional Content Groups

```
cpdf -ocg-list in.pdf
```

List the optional content groups by name.

```
cpdf -ocg-coalesce-on-name in.pdf -o out.pdf
```

Coalesce optional content groups after merging or stamping two files with OCGs with like names.

## Chapter 17: Creating New PDFs

```
cpdf -create-pdf -create-pdf-pages 20  
-create-pdf-papersize usletterportrait -o out.pdf
```

Create a US Letter PDF of twenty pages.

```
cpdf -typeset file.txt -create-pdf-papersize a3portrait  
-font Courier -font-size 10 -o out.pdf
```

Typeset a text file as PDF on A3 paper with Courier 10 point font.

## Chapter 18: Miscellaneous

```
cpdf -blacktext in.pdf -o out.pdf
```

Blacken all the text in in.pdf, writing to out.pdf.

```
cpdf -thinlines 2pt in.pdf -o out.pdf
```

Make sure all lines in in.pdf are at least 2 pts wide, writing to out.pdf.



# Example Program in C#

This program loads a file `hello.pdf` from disk and writes out a document with the original included three times.

```
//Merge example
using System;
using System.Collections.Generic;
using CoherentGraphics;

// Initialise cpdf
Cpdf.startup();

// We will take the input hello.pdf and repeat it three times
using (Cpdf.Pdf mergepdf = Cpdf.fromFile("hello.pdf", ""))
{
    // The list of PDFs to merge
    List<Cpdf.Pdf> pdfs = new List<Cpdf.Pdf> {mergepdf, mergepdf, mergepdf};

    // Merge them
    Cpdf.Pdf merged = Cpdf.mergeSimple(pdfs);

    // Write output
    Cpdf.toFile(merged, "merged.pdf", false, false);

    // Dispose of merged PDF
    merged.Dispose();
}
```

Note the use of `using` and `Dispose()` to ensure the PDFs are thrown away when no longer required.



# Example Program in VB.NET

This program loads a file `hello.pdf` from disk and writes out a document with the original included three times.

```
' Merge example
imports System
imports System.Collections.Generic
imports CoherentGraphics

' Initialise cpdf
Cpdf.startup()

' We will take the input hello.pdf and repeat it three times
Using mergepdf As Cpdf.Pdf = Cpdf.fromFile("hello.pdf", "")
    ' The list of PDFs to merge
    Dim pdfs As List(Of Cpdf.Pdf) =
        new List(Of Cpdf.Pdf)({mergepdf, mergepdf, mergepdf})

    ' Merge them
    Dim merged As Cpdf.Pdf = Cpdf.mergeSimple(pdfs)

    ' Write output
    CpdfToFile(merged, "merged.pdf", false, false)

    ' Dispose of merged PDF
    merged.Dispose()
End Using
```

Note the use of `Using` and `Dispose()` to ensure the PDFs are thrown away when no longer required.



# Contents

<b>1</b>	<b>Basic Usage</b>	<b>1</b>
1.1	Documentation . . . . .	1
1.2	Input and Output Files . . . . .	1
1.3	Input Ranges . . . . .	2
1.4	Working with Encrypted Documents . . . . .	3
1.5	Standard Input and Standard Output . . . . .	4
1.6	Doing Several Things at Once with AND . . . . .	5
1.7	Units . . . . .	5
1.8	Setting the Producer and Creator . . . . .	6
1.9	PDF Version Numbers . . . . .	6
1.10	File IDs . . . . .	6
1.11	Linearization . . . . .	6
1.12	Object Streams . . . . .	7
1.13	Malformed Files . . . . .	8
1.14	Error Handling . . . . .	9
1.15	Control Files . . . . .	9
1.16	String Arguments . . . . .	9
1.17	Text Encodings . . . . .	10
1.18	Font Embedding . . . . .	10
<b>2</b>	<b>Merging and Splitting</b>	<b>19</b>
2.1	Merging . . . . .	19
2.2	Splitting . . . . .	20
2.3	Splitting on Bookmarks . . . . .	20
2.4	Encrypting with Split and Split Bookmarks . . . . .	21
<b>3</b>	<b>Pages</b>	<b>23</b>
3.1	Page Sizes . . . . .	24
3.2	Scale Pages . . . . .	24
3.3	Shift Page Contents . . . . .	25
3.4	Rotating Pages . . . . .	25
3.5	Flipping Pages . . . . .	26
3.6	Boxes and Cropping . . . . .	26
3.7	Showing Boxes and Printer's Marks . . . . .	28

<b>4</b>	<b>Encryption and Decryption</b>	<b>35</b>
4.1	Introduction . . . . .	35
4.2	Encrypting a Document . . . . .	36
4.3	Decrypting a Document . . . . .	36
<b>5</b>	<b>Compression</b>	<b>39</b>
5.1	Decompressing a Document . . . . .	39
5.2	Compressing a Document . . . . .	39
5.3	Squeezing a Document . . . . .	40
<b>6</b>	<b>Bookmarks</b>	<b>43</b>
6.1	List Bookmarks . . . . .	43
6.1.1	Destinations . . . . .	44
6.2	Remove Bookmarks . . . . .	45
6.3	Add Bookmarks . . . . .	45
6.4	Opening bookmarks . . . . .	46
6.5	Making a Table of Contents . . . . .	46
<b>7</b>	<b>Presentations</b>	<b>51</b>
<b>8</b>	<b>Watermarks and Stamps</b>	<b>55</b>
8.1	Add a Watermark or Logo . . . . .	56
8.2	Stamp Text, Dates and Times. . . . .	56
8.2.1	Page Numbers . . . . .	57
8.2.2	Date and Time Formats . . . . .	57
8.2.3	Bates Numbers . . . . .	57
8.2.4	Position . . . . .	58
8.2.5	Font and Size . . . . .	58
8.2.6	Colors . . . . .	59
8.2.7	Outline Text . . . . .	60
8.2.8	Multi-line Text . . . . .	60
8.2.9	Special Characters . . . . .	61
8.3	Stamping Graphics . . . . .	61
8.4	Low-level facilities . . . . .	61
<b>9</b>	<b>Multipage Facilities</b>	<b>67</b>
9.1	Inserting Blank Pages . . . . .	67
9.2	Imposition . . . . .	68
<b>10</b>	<b>Annotations</b>	<b>71</b>
10.1	Listing Annotations . . . . .	71
10.2	Copying Annotations . . . . .	72
10.3	Removing Annotations . . . . .	72



<b>11 Document Information and Metadata</b>	<b>75</b>
11.1 Reading Document Information . . . . .	76
11.2 Setting Document Information . . . . .	77
11.3 XMP Metadata . . . . .	78
11.4 Upon Opening a Document . . . . .	78
11.4.1 Page Layout . . . . .	78
11.4.2 Page Mode . . . . .	79
11.4.3 Display Options . . . . .	80
11.5 Page Labels . . . . .	80
<b>12 File Attachments</b>	<b>91</b>
12.1 Adding Attachments . . . . .	91
12.2 Listing Attachments . . . . .	91
12.3 Removing Attachments . . . . .	92
12.4 Dumping Attachments to File . . . . .	92
<b>13 Working with Images</b>	<b>95</b>
13.1 Extracting images . . . . .	95
13.2 Detecting Low-resolution Images . . . . .	95
13.3 Removing an Image . . . . .	96
<b>14 Fonts</b>	<b>99</b>
14.1 Listing Fonts . . . . .	99
14.2 Listing characters in a font . . . . .	100
14.3 Copying Fonts . . . . .	101
14.4 Removing Fonts . . . . .	101
14.5 Missing Fonts . . . . .	101
<b>15 PDF and JSON</b>	<b>105</b>
15.1 Converting PDF to JSON . . . . .	105
15.2 Converting JSON to PDF . . . . .	107
<b>16 Optional Content Groups</b>	<b>109</b>
<b>17 Creating New PDFs</b>	<b>111</b>
17.1 A new blank PDF . . . . .	111
17.2 Convert a text file to PDF . . . . .	111
<b>18 Miscellaneous</b>	<b>113</b>
18.1 Draft Documents . . . . .	113
18.2 Blackening Text, Lines and Fills . . . . .	114
18.3 Hairline Removal . . . . .	114
18.4 Garbage Collection . . . . .	115
18.5 Change PDF Version Number . . . . .	115
18.6 Copy ID . . . . .	115
18.7 Remove ID . . . . .	115
18.8 List Spot Colours . . . . .	116
18.9 PDF Dictionary Entries . . . . .	116

18.10 Removing Clipping . . . . .	117
<b>A Dates</b>	<b>121</b>
A.1 PDF Date Format . . . . .	121
A.2 XMP Metadata Date Format . . . . .	122

# Typographical Conventions

Command lines to be typed are shown in typewriter font in a box. For example:

```
cpdf in.pdf -o out.pdf
```

When describing the general form of a command, rather than a particular example, square brackets [] are used to enclose optional parts, and angled braces <> to enclose general descriptions which may be substituted for particular instances. For example,

```
cpdf <operation> in.pdf [<range>] -o out.pdf
```

describes a command line which requires an operation and, optionally, a range. An exception is that we use `in.pdf` and `out.pdf` instead of `<input file>` and `<output file>` to reduce verbosity. Under Microsoft Windows, type `cpdf.exe` instead of `cpdf`.



# Chapter 1

## Basic Usage

```
-help          --help          -version
-o            -i              -idir <directory>
-encrypt      -decrypt-force -stdout
-stdin        -stdin-user <password> -stdin-owner <password>
-producer <text> -creator <text> -change-id
-l           -cpdfin <filename> -keep-l
-no-preserve-objstm -create-objstm -control <filename>
-args <filename> -utf8 -stripped
-raw         -no-embed-font -gs
-gs-malformed -gs-malformed-force -gs-quiet
-error-on-malformed
```

The Coherent PDF tools provide a wide range of facilities for modifying PDF files created by other means. There is a single command-line program `cpdf` (`cpdf.exe` under Microsoft Windows). The rest of this manual describes the options that may be given to this program.

### 1.1 Documentation

The operation `-help / --help` prints each operation and option together with a short description. The operation `-version` prints the `cpdf` version string.

### 1.2 Input and Output Files

The typical pattern for usage is

```
cpdf [<operation>] <input file(s)> -o <output file>
```

and the simplest concrete example, assuming the existence of a file `in.pdf` is:

```
cpdf in.pdf -o out.pdf
```

which copies `in.pdf` to `out.pdf`. The input and output may be the same file. Of course, we should like to do more interesting things to the PDF file than that!

Files on the command line are distinguished from other input by their containing a period. If an input file does not contain a period, it should be preceded by `-i`. For example:

```
cpdf -i in -o out.pdf
```

A whole directory of files may be added (where a command supports multiple files) by using the `-idir` option:

```
cpdf -merge -idir myfiles -o out.pdf
```

The files in the directory `myfiles` are considered in alphabetical order. They must all be PDF files. If the names of the files are numeric, leading zeroes will be required for the order to be correct (e.g. `001.pdf`, `002.pdf` etc).

To restrict `cpdf` to files ending in `.pdf` (in upper or lower or mixed case) add the option `-idir-only-pdfs` *before* `-idir`:

```
cpdf -merge -idir-only-pdfs -idir myfiles -o out.pdf
```

## 1.3 Input Ranges

An *input range* may be specified after each input file. This is treated differently by each operation. For instance

```
cpdf in.pdf 2-5 -o out.pdf
```

extracts pages two, three, four and five from `in.pdf`, writing the result to `out.pdf`, assuming that `in.pdf` contains at least five pages. Here are the rules for building input ranges:

- A dash (`-`) defines ranges, e.g. `1-5` or `6-3`.
- A comma (`,`) allows one to specify several ranges, e.g. `1-2, 4-5`.
- The word `end` represents the last page number.
- The words `odd` and `even` can be used in place of or at the end of a page range to restrict to just the odd or even pages.

- The words `portrait` and `landscape` can be used in place of or at the end of a page range to restrict to just those pages which are portrait or landscape. Note that the meaning of “portrait” and “landscape” does not take account of any viewing rotation in place (use `-upright` from chapter 3 first, if required). A page with equal width and height is considered neither portrait nor landscape.
- The word `reverse` is the same as `end-1`.
- The word `all` is the same as `1-end`.
- A range must contain no spaces.
- A tilde (`~`) defines a page number counting from the end of the document rather than the beginning. Page `~1` is the last page, `~2` the penultimate page etc.
- Prepending `NOT` to a whole page range inverts it.
- Prepending `<n>DUP` to a whole page range duplicates each page of the range `<n>` times.

For example:

```

cpdf in.pdf 1,2,7-end -o out.pdf
Remove pages three, four, five and six from a document.

cpdf in.pdf 1-16odd -o out.pdf
Extract the odd pages 1,3,...,13,15.

cpdf in.pdf landscape -rotate 90 -o out.pdf
Rotate all landscape pages by ninety degrees.

cpdf in.pdf 1,all -o out.pdf
Duplicate the front page of a document, perhaps as a fax cover sheet.

cpdf in.pdf ~3-~1 -o out.pdf
Extract the last three pages of a document, in order.

cpdf in.pdf 2DUP1-10 -o out.pdf
Produce the pages 1,1,2,2,...,10,10.

```

## 1.4 Working with Encrypted Documents

In order to perform many operations, encrypted input PDF files must be decrypted. Some require the owner password, some either the user or owner passwords. Either password is supplied by writing `user=<password>` or `owner=<password>` following each input file requiring it (before or after any range). The document will *not* be re-encrypted upon writing. For example:

```
cpdf in.pdf user=charles -info
cpdf in.pdf owner=fred reverse -o out.pdf
```

To re-encrypt the file with its existing encryption upon writing, which is required if only the user password was supplied, but allowed in any case, add the `-recrypt` option:

```
cpdf in.pdf user=charles reverse -recrypt -o out.pdf
```

The password required (owner or user) depends upon the operation being performed. Separate facilities are provided to decrypt and encrypt files (See Section 4).

When appropriate passwords are not available, the option `-decrypt-force` may be added to the command line to process the file regardless.

## 1.5 Standard Input and Standard Output

Thus far, we have assumed that the input PDF will be read from a file on disk, and the output written similarly. Often it's useful to be able to read input from `stdin` (Standard Input) or write output to `stdout` (Standard Output) instead. The typical use is to join several programs together into a *pipe*, passing data from one to the next without the use of intermediate files. Use `-stdin` to read from standard input, and `-stdout` to write to standard output, either to pipe data between multiple programs, or multiple invocations of the same program. For example, this sequence of commands (all typed on one line)

```
cpdf in.pdf reverse -stdout |
cpdf -stdin 1-5 -stdout |
cpdf -stdin reverse -o out.pdf
```

extracts the last five pages of `in.pdf` in the correct order, writing them to `out.pdf`. It does this by reversing the input, taking the first five pages and then reversing the result.

To supply passwords for a file from `-stdin`, use `-stdin-owner <password>` and/or `-stdin-user <password>`.

Using `-stdout` on the final command in the pipeline to output the PDF to screen is not recommended, since PDF files often contain compressed sections which are not screen-readable.

Several `cpdf` operations write to standard output by default (for example, listing fonts). A useful feature of the command line (not specific to `cpdf`) is the ability to redirect this output to a file. This is achieved with the `>` operator:

```
cpdf -info in.pdf > file.txt
```

Use the `-info` operation (See Section 11.1), redirecting the output to `file.txt`.



## 1.6 Doing Several Things at Once with AND

The keyword `AND` can be used to string together several commands in one. The advantage compared with using pipes is that the file need not be repeatedly parsed and written out, saving time.

To use `AND`, simply leave off the output specifier (e.g. `-o`) of one command, and the input specifier (e.g. filename) of the next. For instance:

```
cpdf -merge in.pdf in2.pdf AND -add-text "Label"
      AND -merge in3.pdf -o out.pdf
```

Merge `in.pdf` and `in2.pdf` together, add text to both pages, append `in3.pdf` and write to `out.pdf`.

To specify the range for each section, use `-range`:

```
cpdf -merge in.pdf in2.pdf AND -range 2-4 -add-text "Label"
      AND -merge in3.pdf -o out.pdf
```

## 1.7 Units

When measurements are given to `cpdf`, they are in points (1 point = 1/72 inch). They may optionally be followed by some letters to change the measurement. The following are supported:

<code>pt</code>	Points (72 points per inch). The default.
<code>cm</code>	Centimeters
<code>mm</code>	Millimeters
<code>in</code>	Inches

For example, one may write `14mm` or `21.6in`. In addition, the following letters stand, in some operations (`-scale-page`, `-scale-to-fit`, `-scale-contents`, `-shift`, `-mediabox`, `-crop`) for various page dimensions:

<code>PW</code>	Page width
<code>PH</code>	Page height
<code>PMINX</code>	Page minimum x coordinate
<code>PMINY</code>	Page minimum y coordinate
<code>PMAXX</code>	Page maximum x coordinate
<code>PMAXY</code>	Page maximum y coordinate
<code>CW</code>	Crop box width
<code>CH</code>	Crop box height
<code>CMINX</code>	Crop box minimum x coordinate
<code>CMINY</code>	Crop box minimum y coordinate
<code>CMAXX</code>	Crop box maximum x coordinate
<code>CMAXY</code>	Crop box maximum y coordinate

For example, we may write `PMINX PMINY` to stand for the coordinate of the lower left corner of the page.

Simple arithmetic may be performed using the words `add`, `sub`, `mul` and `div` to stand for addition, subtraction, multiplication and division. For example, one may write `14in sub 30pt` or `PMINX mul 2`

## 1.8 Setting the Producer and Creator

The `-producer` and `-creator` options may be added to any `cpdf` command line to set the producer and/or creator of the PDF file. If the file was converted from another format, the *creator* is the program producing the original, the *producer* the program converting it to PDF.

```
cpdf -merge in.pdf in2.pdf -producer MyMerger -o out.pdf
```

Merge `in.pdf` and `in2.pdf`, setting the producer to `MyMerger` and writing the output to `out.pdf`.

## 1.9 PDF Version Numbers

When an operation which uses a part of the PDF standard which was introduced in a later version than that of the input file, the PDF version in the output file is set to the later version (most PDF viewers will try to load any PDF file, even if it is marked with a later version number). However, this automatic version changing may be suppressed with the `-keep-version` option. If you wish to manually alter the PDF version of a file, use the `-set-version` operation described in Section 18.5.

## 1.10 File IDs

PDF files contain an ID (consisting of two parts), used by some workflow systems to uniquely identify a file. To change the ID, behavior, use the `-change-id` operation. This will create a new ID for the output file.

```
cpdf -change-id in.pdf -o out.pdf
```

Write `in.pdf` to `out.pdf`, changing the ID.

## 1.11 Linearization

Linearized PDF is a version of the PDF format in which the data is held in a special manner to allow content to be fetched only when needed. This means viewing a multipage PDF over a slow connection is more responsive. By default, `cpdf` does not linearize output files. To make it

do so, add the `-l` option to the command line, in addition to any other command being used. For example:

```
cpdf -l in.pdf -o out.pdf
```

Linearize the file `in.pdf`, writing to `out.pdf`.

This requires the existence of the external program `cpdfwin` which is provided with commercial versions of `cpdf`. This must be installed as described in the installation documentation provided with your copy of `cpdf`. If you are unable to install `cpdfwin`, you must use `-cpdfwin` to let `cpdf` know where to find it:

```
cpdf.exe -cpdfwin "C:\\cpdfwin.exe" -l in.pdf -o out.pdf
```

Linearize the file `in.pdf`, writing to `out.pdf`.

In extremis, you may place `cpdfwin` and its resources in the current working directory, though this is not recommended. For further help, refer to the installation instructions for your copy of `cpdf`.

To keep the existing linearization status of a file (produce linearized output if the input is linearized and the reverse), use `-keep-l` instead of `-l`.

## 1.12 Object Streams

PDF 1.5 introduced a new mechanism for storing objects to save space: object streams. by default, `cpdf` will preserve object streams in input files, creating no more. To prevent the retention of existing object streams, use `-no-preserve-objstm`:

```
cpdf -no-preserve-objstm in.pdf -o out.pdf
```

Write the file `in.pdf` to `out.pdf`, removing any object streams.

To create new object streams if none exist, or augment the existing ones, use `-create-objstm`:

```
cpdf -create-objstm in.pdf -o out.pdf
```

Write the file `in.pdf` to `out.pdf`, preserving any existing object streams, and creating any new ones for new objects which have been added.

To create wholly new object streams, use both options together:

```
cpdf -create-objstm -no-preserve-objstm in.pdf -o out.pdf
```

Write the file `in.pdf` to `out.pdf` with wholly new object streams.

Files written with object streams will be set to PDF 1.5 or higher, unless `-keep-version` is used (see above).

## 1.13 Malformed Files

There are many malformed PDF files in existence, including many produced by otherwise-reputable applications. `cpdf` attempts to correct these problems silently.

Grossly malformed files will be reconstructed. The reconstruction progress is shown on `stderr` (Standard Error):

```
$cpdf in.pdf -o out.pdf
couldn't lex object number
Attempting to reconstruct the malformed pdf in.pdf...
Read 5530 objects
Malformed PDF reconstruction succeeded!
```

If `cpdf` cannot reconstruct a malformed file, it is able to use the `gs` program to try to reconstruct the PDF file, if you have it installed. For example, if `gs` is installed and in your path, we might try:

```
cpdf -gs gs -gs-malformed in.pdf -o out.pdf
```

To suppress the output of `gs` use the `-gs-quiet` option.

If the malformity lies inside an individual page of the PDF, rather than in its gross structure, `cpdf` may appear to succeed in reconstruction, only to fail when processing a page (e.g when adding text). To force the use of `gs` to pre-process such files so `cpdf` cannot fail on them, use `-gs-malformed-force`:

```
cpdf in.pdf -gs gs -gs-malformed-force -o out.pdf [-gs-quiet]
```

The command line for `-gs-malformed-force` must be of *precisely* this form. Sometimes, on the other hand, we might wish `cpdf` to fail immediately on any malformed file, rather than try its own reconstruction process. The option `-error-on-malformed` achieves this.

Sometimes (old, pre-ISO standardisation) files can be technically well-formed but use inefficient PDF constructs. If you are sure the input files you are using are well formed, the `-fast` option may be added to the command line (or, if using AND, to each section of the command line). This will use certain shortcuts which speed up processing, but would fail on badly-produced files. The `-fast` option may be used with:

```
Chapter 3
-rotate-contents -upright -vflip -hflip
-shift -scale-page -scale-to-fit -scale-contents
```

```
-show-boxes -hard-box -trim-marks  
  
Chapter 8  
-add-text -add-rectangle  
-stamp-on -stamp-under -combine-pages  
  
Chapter 9  
-impose -impose-xy -twoup -twoup-stack
```

If problems occur, refrain from using `-fast`.

## 1.14 Error Handling

When `cpdf` encounters an error, it exits with code 2. An error message is displayed on `stderr` (Standard Error). In normal usage, this means it's displayed on the screen. When a bad or inappropriate password is given, the exit code is 1.

## 1.15 Control Files

```
cpdf -control <filename>  
cpdf -args <filename>
```

Some operating systems have a limit on the length of a command line. To circumvent this, or simply for reasons of flexibility, a control file may be specified from which arguments are drawn. This file does not support the full syntax of the command line. Commands are separated by whitespace, quotation marks may be used if an argument contains a space, and the sequence `\ "` may be used to introduce a genuine quotation mark in such an argument.

Several `-control` arguments may be specified, and may be mixed in with conventional command-line arguments. The commands in each control file are considered in the order in which they are given, after all conventional arguments have been processed. It is recommended to use `-args` in all new applications. However, `-control` will be supported for legacy applications.

To avoid interference between `-control` and `AND`, a new mechanism has been added. Using `-args` in place of `-control` will perform direct textual substitution of the file into the command line, prior to any other processing.

## 1.16 String Arguments

Command lines are handled differently on each operating system. Some characters are reserved with special meanings, even when they occur inside quoted string arguments. To avoid this problem, `cpdf` performs processing on string arguments as they are read.

A backslash is used to indicate that a character which would otherwise be treated specially by the command line interpreter is to be treated literally. For example, Unix-like systems attribute a special meaning to the exclamation mark, so the command line

```
cpdf -add-text "Hello!" in.pdf -o out.pdf
```

would fail. We must escape the exclamation mark with a backslash:

```
cpdf -add-text "Hello\!" in.pdf -o out.pdf
```

It follows that backslashes intended to be taken literally must themselves be escaped (i.e. written `\\`).

## 1.17 Text Encodings

Some `cpdf` commands write text to standard output, or read text from the command line or configuration files. These are:

```
-info  
-list-bookmarks  
-set-author et al.  
-list-annotations  
-dump-attachments
```

There are three options to control how the text is interpreted:

```
-utf8  
-stripped  
-raw
```

Add `-utf8` to use Unicode UTF8, `-stripped` to convert to 7 bit ASCII by dropping any high characters, or `-raw` to perform no processing. The default unless specified in the documentation for an individual operation is `-stripped`.

## 1.18 Font Embedding

Use the `-no-embed-font` to avoid embedding the Standard 14 Font metrics when adding text with `-add-text`.

## .NET Interface

CPDF .NET Library  
=====

Before using the library, you must make sure your project or build environment has access to the cpdf DLL, which is not part of the .NET assembly. You can add it to a Visual Studio project as a file, set to copy-to-output-folder. Or, you can install it in a standard location. The DLL must be named as follows, for .NET to be able to find it:

Windows: cpdf.dll  
MacOS: libcpdf.dylib  
Linux: libcpdf.so

The following documentation is automatically extracted from the XML comments present in the assembly. You can use Intellisense to view them within Visual Studio - the documentation here is for completeness.

A 'range' is a list of integers specifying page numbers.

Text arguments and results are in UTF8. Data is of type bytes[].

Any function may raise the exception CPDFError, carrying a string describing the error.

There is no automatic garbage collection of PDFs. Use the Using keyword and/or the Dispose method on objects of type Cpdf.Pdf to explicitly deallocate PDFs.

### CHAPTER 0. Preliminaries

Cpdf.startup

Initialises the library. Must be called before any other function.

Cpdf.version

Returns a string giving the version number of the CPDF library.

Cpdf.setFast

Some operations have a fast mode. The default is 'slow' mode, which works even on old-fashioned files. For more details, see section 1.13 of the CPDF manual. This function sets the mode to fast globally.

Cpdf.setSlow

Some operations have a fast mode. The default is 'slow' mode, which works even on old-fashioned files. For more details, see section 1.13 of the CPDF manual. This functions sets the mode to slow globally.

`Cpdf.lastError`

Not to be called directly. Errors in .NET cpdf are raised by exceptions.

`Cpdf.lastErrorString`

Not to be called directly. Errors in .NET cpdf are raised by exceptions.

`Cpdf.clearError`

Not to be called directly. Errors in .NET cpdf are raised by exceptions.

`Cpdf.onExit`

A debug function which prints some information about resource usage. This can be used to detect if PDFs or ranges are being deallocated properly. Contrary to its name, it may be run at any time.

## CHAPTER 1. Basics

`Cpdf.fromFile(String, String)`

Loads a PDF file from a given file. Supply a user password (possibly blank) in case the file is encrypted. It won't be decrypted, but sometimes the password is needed just to load the file.

`Cpdf.fromFileLazy(String, String)`

Loads a PDF from a file, doing only minimal parsing. The objects will be read and parsed when they are actually needed. Use this when the whole file won't be required. Also supply a user password (possibly blank) in case the file is encrypted. It won't be decrypted, but sometimes the password is needed just to load the file.

`Cpdf.fromMemory(Byte[], String)`

Loads a file from memory given any user password.

`Cpdf.fromMemoryLazy(IntPtr, Int32, String)`

Loads a file from memory, given a pointer and a length, and the user password, but lazily like `fromFileLazy`. The caller must use `AllocHGlobal / Marshal.Copy / FreeHGlobal` itself. It must not free the memory until the PDF is also gone.

`Cpdf.startEnumeratePDFs`

`Cpdf.enumeratePDFsKey(Int32)`

`Cpdf.enumeratePDFsInfo(Int32)`



`Cpdf.endEnumeratePDFs`

To enumerate the list of currently allocated PDFs, call `startEnumeratePDFs` which gives the number, `n`, of PDFs allocated, then `enumeratePDFsInfo` and `enumeratePDFsKey` with index numbers from `0...(n - 1)`. Call `endEnumeratePDFs` to clean up.

`Cpdf.ptOfCm(Double)`

Converts a figure in centimetres to points (72 points to 1 inch)

`Cpdf.ptOfMm(Double)`

Converts a figure in millimetres to points (72 points to 1 inch)

`Cpdf.ptOfIn(Double)`

Converts a figure in inches to points (72 points to 1 inch)

`Cpdf.cmOfPt(Double)`

Converts a figure in points to centimetres (72 points to 1 inch)

`Cpdf.mmOfPt(Double)`

Converts a figure in points to millimetres (72 points to 1 inch)

`Cpdf.inOfPt(Double)`

Converts a figure in points to inches (72 points to 1 inch)

`Cpdf.parsePagespec(Cpdf.Pdf, String)`

Parses a page specification with reference to a given PDF (the PDF is supplied so that page ranges which reference pages which do not exist are rejected).

`Cpdf.validatePagespec(String)`

Validates a page specification so far as is possible in the absence of the actual document. Result is true if valid.

`Cpdf.stringOfPagespec(Cpdf.Pdf, List{Int32})`

Builds a page specification from a page range. For example, the range containing 1, 2, 3, 6, 7, 8 in a document of 8 pages might yield "1-3, 6-end"

`Cpdf.blankRange`

Creates a range with no pages in.

`Cpdf.range(Int32, Int32)`

Builds a range from one page to another inclusive. For example, `range(3, 7)` gives the range 3, 4, 5, 6, 7

```
Cpdf.all(Cpdf.Pdf)
```

The range containing all the pages in a given document.

```
Cpdf.even(List{Int32})
```

Makes a range which contains just the even pages of another range.

```
Cpdf.odd(List{Int32})
```

Makes a range which contains just the odd pages of another range.

```
Cpdf.rangeUnion(List{Int32}, List{Int32})
```

Makes the union of two ranges giving a range containing the pages in range a and range b.

```
Cpdf.difference(List{Int32}, List{Int32})
```

Makes the difference of two ranges, giving a range containing all the pages in a except for those which are also in b.

```
Cpdf.removeDuplicates(List{Int32})
```

Deduplicates a range, making a new one.

```
Cpdf.rangeLength(List{Int32})
```

Gives the number of pages in a range.

```
Cpdf.rangeGet(List{Int32}, Int32)
```

Gets the page number at position n in a range, where n runs from 0 to `rangeLength - 1`.

```
Cpdf.rangeAdd(List{Int32}, Int32)
```

Adds the page to a range, if it is not already there.

```
Cpdf.isInRange(List{Int32}, Int32)
```

Returns true if the page is in the range, false otherwise.

```
Cpdf.pages(Cpdf.Pdf)
```

Returns the number of pages in a PDF.

`Cpdf.pagesFast(String, String)`

Returns the number of pages in a given PDF, with given user password. It tries to do this as fast as possible, without loading the whole file.

`Cpdf.toFile(Cpdf.Pdf, String, Boolean, Boolean)`

Writes the file to a given filename. If `linearize` is true, it will be linearized if a linearizer is available. If `make_id` is true, it will be given a new ID.

`Cpdf.toFileExt(Cpdf.Pdf, String, Boolean, Boolean, Boolean, Boolean, Boolean)`

Writes the file to a given filename. If `make_id` is true, it will be given a new ID. If `preserve_objstm` is true, existing object streams will be preserved. If `generate_objstm` is true, object streams will be generated even if not originally present. If `compress_objstm` is true, object streams will be compressed (what we usually want). **WARNING:** the `pdf` argument will be invalid after this call, and should be not be used again.

`Cpdf.toMemory(Cpdf.Pdf, Boolean, Boolean)`

Writes a PDF file and returns as an array of bytes.

`Cpdf.isEncrypted(Cpdf.Pdf)`

Returns true if a documented is encrypted, false otherwise.

`Cpdf.decryptPdf(Cpdf.Pdf, String)`

Attempts to decrypt a PDF using the given user password. An exception is raised if the decryption fails.

`Cpdf.decryptPdfOwner(Cpdf.Pdf, String)`

Attempts to decrypt a PDF using the given owner password. Raises an exception if the decryption fails.

`Cpdf.Permission`

Permissions

`Cpdf.Permission.NoEdit`

Cannot edit the document

`Cpdf.Permission.NoPrint`

Cannot print the document

`Cpdf.Permission.NoCopy`

Cannot copy the document

`Cpdf.Permission.NoAnnot`

Cannot annotate the document

`Cpdf.Permission.NoForms`

Cannot edit forms in the document

`Cpdf.Permission.NoExtract`

Cannot extract information

`Cpdf.Permission.NoAssemble`

Cannot assemble into a bigger document

`Cpdf.Permission.NoHqPrint`

Cannot print high quality

`Cpdf.EncryptionMethod`

Encryption methods

`Cpdf.EncryptionMethod.Pdf40bit`

40 bit RC4 encryption

`Cpdf.EncryptionMethod.Pdf128bit`

128 bit RC4 encryption

`Cpdf.EncryptionMethod.Aes128bitfalse`

128 bit AES encryption, do not encrypt metadata

`Cpdf.EncryptionMethod.Aes128bittrue`

128 bit AES encryption, encrypt metadata

`Cpdf.EncryptionMethod.Aes256bitfalse`

Deprecated. Do not use for new files

`Cpdf.EncryptionMethod.Aes256bittrue`

Deprecated. Do not use for new files

`Cpdf.EncryptionMethod.Aes256bitisofalse`

256 bit AES encryption, do not encrypt metadata

`Cpdf.EncryptionMethod.Aes256bitiosottrue`

256 bit AES encryption, encrypt metadata

`Cpdf.toFileEncrypted(Cpdf.Pdf, Cpdf.EncryptionMethod, List{Cpdf.Permission}, String, String, Boolean, Boolean, String)`

Writes a file as encrypted.

`Cpdf.toFileEncryptedExt(Cpdf.Pdf, Cpdf.EncryptionMethod, List{Cpdf.Permission}, String, String, Boolean, Boolean, Boolean, Boolean, Boolean, String)`

Writes a file as encrypted with extra parameters. WARNING: the pdf argument will be invalid after this call, and should not be used again.

`Cpdf.hasPermission(Cpdf.Pdf, Cpdf.Permission)`

Returns true if the given permission (restriction) is present.

`Cpdf.encryptionKind(Cpdf.Pdf)`

Returns the encryption method currently in use on a document.



## Chapter 2

# Merging and Splitting

```
cpdf -merge in1.pdf [<range>] in2.pdf [<range>] [<more names/ranges>]
    [-collate] [-retain-numbering] [-remove-duplicate-fonts]
    [-merge-add-bookmarks [-merge-add-bookmarks-use-titles]]
    -o out.pdf

cpdf -split in.pdf -o <format> [-chunk <chunksize>]
cpdf -split-bookmarks <level> in.pdf -o <format>
```

## 2.1 Merging

The `-merge` operation allow the merging of several files into one. Ranges can be used to select only a subset of pages from each input file in the output. The output file consists of the concatenation of all the input pages in the order specified on the command line. Actually, the `-merge` can be omitted, since this is the default operation of `cpdf`.

```
cpdf -merge a.pdf 1 b.pdf 2-end -o out.pdf
```

Take page one of `a.pdf` and all but the first page of `b.pdf`, merge them and produce `out.pdf`.

```
cpdf -merge -idir files -o out.pdf
```

Merge all files from directory `files`, producing `out.pdf`.

Merge maintains bookmarks, named destinations, and name dictionaries.

PDF features which cannot be merged are retained if they are from the document which first exhibits that feature.

The `-collate` option collates pages: that is to say, it takes the first page from the first document and its range, then the first page from the second document and its range and so on. When all first pages have been taken, it begins on the second from each range, and so on.

The `-retain-numbering` option keeps the PDF page numbering labels of each document intact, rather than renumbering the output pages from 1.

The `-remove-duplicate-fonts` option ensures that fonts used in more than one of the inputs only appear once in the output.

The `-merge-add-bookmarks` option adds a top-level bookmark for each file, using the filename. Any existing bookmarks are retained. The `-merge-add-bookmarks-use-titles`, when used in conjunction with `-merge-add-bookmarks`, will use the title from each PDF's metadata instead of the filename.

## 2.2 Splitting

The `-split` operation splits a PDF file into a number of parts which are written to file, their names being generated from a *format*. The optional `-chunk` option allows the number of pages written to each output file to be set.

```
cpdf -split a.pdf -o out%%%.pdf
```

Split a.pdf to the files out001.pdf, out002.pdf etc.

```
cpdf a.pdf even AND -split -chunk 10 -o dir/out%%%.pdf
```

Split the even pages of a.pdf to the files out001.pdf, out002.pdf etc. with at most ten pages in each file. The directory (folder) dir must exist.

If the output format does not provide enough numbers for the files generated, the result is unspecified. The following format operators may be used:

<code>%</code> , <code>%%</code> , <code>%%%</code> etc.	Sequence number padded to the number of percent signs
<code>@F</code>	Original filename without extension
<code>@N</code>	Sequence number without padding zeroes
<code>@S</code>	Start page of this chunk
<code>@E</code>	End page of this chunk
<code>@B</code>	Bookmark name at this page

Numbers padded to a fixed width field by zeroes may be obtained for `@S` and `@E` by following them with more `@` signs e.g. `@E@@@` for a fixed width of three.

## 2.3 Splitting on Bookmarks

The `-split-bookmarks <level>` operation splits a PDF file into a number of parts, according to the page ranges implied by the document's bookmarks. These parts are then written to file with names generated from the given format.

Level 0 denotes the top-level bookmarks, level 1 the next level (sub-bookmarks) and so on. So `-split-bookmarks 1` creates breaks on level 0 and level 1 boundaries.



```
cpdf -split-bookmarks 0 a.pdf -o out%%%.pdf
```

Split `a.pdf` to the files `out001.pdf`, `out002.pdf` on bookmark boundaries.

Now, there may be many bookmarks on a single page (for instance, if paragraphs are bookmarked or there are two subsections on one page). The splits calculated by `-split-bookmarks` ensure that each page appears in only one of the output files. It is possible to use the `@` operators above, including operator `@B` which expands to the text of the bookmark:

```
cpdf -split-bookmarks 0 a.pdf -o @B.pdf
```

Split `a.pdf` on bookmark boundaries, using the bookmark text as the filename.

The bookmark text used for a name is converted from unicode to 7 bit ASCII, and the following characters are removed, in addition to any character with ASCII code less than 32:

```
/ ? < > \ : * | " ^ + =
```

## 2.4 Encrypting with Split and Split Bookmarks

The encryption parameters described in Chapter 4 may be added to the command line to encrypt each split PDF. Similarly, the `-reencrypt` switch described in Chapter 1 may be given to re-encrypt each file with the existing encryption of the source PDF.

## .NET Interface

CHAPTER 2. Merging and Splitting

```
Cpdf.mergeSimple(List{Cpdf.Pdf})
```

Given a list of PDFs, merges the files into a new one, which is returned.

```
Cpdf.merge(List{Cpdf.Pdf}, Boolean, Boolean)
```

Merges the PDFs. If `retain_numbering` is true page labels are not rewritten. If `remove_duplicate_fonts` is true, duplicate fonts are merged. This is useful when the source documents for merging originate from the same source.

```
Cpdf.mergeSame(List{Cpdf.Pdf}, Boolean, Boolean, List{List{Int32}})
```

The same as `merge`, except that it has an additional argument - a list of page ranges. This is used to select the pages to pick from each PDF. This avoids duplication of information when multiple discrete parts of a source PDF are included.

```
Cpdf.selectPages(Cpdf.Pdf, List{Int32})
```

Returns a new document which just those pages in the page range.

# Chapter 3

## Pages

```
cpdf -scale-page "<scale x> <scale y>" [-fast] in.pdf [<range>] -o out.pdf
cpdf -scale-to-fit "<x size> <y size>" [-fast]
  [-scale-to-fit-scale <scale>]
  in.pdf [<range>] -o out.pdf
cpdf -scale-contents [<scale>] [<position>] [-fast]
  in.pdf [<range>] -o out.pdf
cpdf -shift "<shift x> <shift y>" [-fast] in.pdf [<range>] -o out.pdf
cpdf -rotate <angle> in.pdf [<range>] -o out.pdf
cpdf -rotateby <angle> in.pdf [<range>] -o out.pdf
cpdf -upright [-fast] in.pdf [<range>] -o out.pdf
cpdf -rotate-contents <angle> [-fast] in.pdf [<range>] -o out.pdf
cpdf -hflip [-fast] in.pdf [<range>] -o out.pdf
cpdf -vflip [-fast] in.pdf [<range>] -o out.pdf
cpdf -mediabox "<x> <y> <w> <h>" in.pdf [<range>] -o out.pdf
cpdf -cropbox "<x> <y> <w> <h>" in.pdf [<range>] -o out.pdf
cpdf -remove-cropbox in.pdf [<range>] -o out.pdf
```

(Also bleed, art, and trim versions of these two commands, for example `-artbox`, `-remove-trimbox`)

```
cpdf -frombox <boxname> -tobox <boxname> [-mediabox-if-missing]
  in.pdf [<range>] -o out.pdf
cpdf -hard-box <boxname> [-fast] in.pdf [<range>]
  [-mediabox-if-missing] -o out.pdf
cpdf -show-boxes [-fast] in.pdf [<range>] -o out.pdf
cpdf -trim-marks [-fast] in.pdf [<range>] -o out.pdf
```

### 3.1 Page Sizes

Any time when a page size is required, instead of writing, for instance "210mm 197mm" one can instead write a4portrait. Here is a list of supported page sizes:

a0portrait	a1portrait	a2portrait
a3portrait	a4portrait	a5portrait
a6portrait	a7portrait	a8portrait
a9portrait	a10portrait	
a0landscape	a1landscape	a2landscape
a3landscape	a4landscape	a5landscape
a6landscape	a7landscape	a8landscape
a9landscape	a10landscape	
usletterportrait	usletterlandscape	
uslegalportrait	uslegallandscape	

### 3.2 Scale Pages

The `-scale-page` operation scales each page in the range by the X and Y factors given. This scales both the page contents, and the page size itself. It also scales any Crop Box and other boxes (Art Box, Trim Box etc). As with several of these commands, remember to take into account any page rotation when considering what the X and Y axes relate to.

```
cpdf -scale-page "2 2" in.pdf -o out.pdf
```

Convert an A4 page to A3, for instance.

The `-scale-to-fit` operation scales each page in the range to fit a given page size, preserving aspect ratio and centering the result.

```
cpdf -scale-to-fit "297mm 210mm" in.pdf -o out.pdf
cpdf -scale-to-fit a4portrait in.pdf -o out.pdf
```

Scale a file's pages to fit A4 portrait.

The scale can optionally be set to a percentage of the available area, instead of filling it.

```
cpdf -scale-to-fit a4portrait -scale-to-fit-scale 0.9 in.pdf -o out.pdf
```

Scale a file's pages to fit A4 portrait, scaling the page 90% of its possible size.

NB: `-scale-to-fit` operates with respect to the media box not the crop box. If necessary, set the media box to be equal to the crop box first. In addition, `-scale-to-fit` presently requires

that the origin of the media box be (0, 0). This can be assured by preprocessing with `-upright` (described elsewhere in this chapter).

The `-scale-contents` operation scales the contents about the center of the crop box (or, if absent, the media box), leaving the page dimensions (boxes) unchanged.

```
cpdf -scale-contents 0.5 in.pdf -o out.pdf
```

Scale a file's contents on all pages to 50% of its original dimensions.

To scale about a point other than the center, one can use the positioning commands described in Section 8.2.4. For example:

```
cpdf -scale-contents 0.5 -topright 20 in.pdf -o out.pdf
```

Scale a file's contents on all pages to 50% of its original dimensions about a point 20pts from its top right corner.

### 3.3 Shift Page Contents

The `-shift` operation shifts the contents of each page in the range by X points horizontally and Y points vertically.

```
cpdf -shift "50 0" in.pdf even -o out.pdf
```

Shift pages to the right by 50 points (for instance, to increase the binding margin).

### 3.4 Rotating Pages

There are two ways of rotating pages: (1) setting a value in the PDF file which asks the viewer (e.g. Acrobat) to rotate the page on-the-fly when viewing it (use `-rotate` or `-rotateby`) and (2) actually rotating the page contents and/or the page dimensions (use `-upright` (described elsewhere in this chapter) afterwards or `-rotate-contents` to just rotate the page contents).

The possible values for `-rotate` and `-rotate-by` are 0, 90, 180 and 270, all interpreted as being clockwise. Any value may be used for `-rotate-contents`.

The `-rotate` operation sets the viewing rotation of the selected pages to the absolute value given.

```
cpdf -rotate 90 in.pdf -o out.pdf
```

Set the rotation of all the pages in the input file to ninety degrees clockwise.

The `-rotateby` operation changes the viewing rotation of all the given pages by the relative value given.

```
cpdf -rotateby 90 in.pdf -o out.pdf
```

Rotate all the pages in the input file by ninety degrees clockwise.

The `-rotate-contents` operation rotates the contents and dimensions of the page by the given relative value.

```
cpdf -rotate-contents 90 in.pdf -o out.pdf
```

Rotate all the page contents in the input file by ninety degrees clockwise. Does not change the page dimensions.

The `-upright` operation does whatever combination of `-rotate` and `-rotate-contents` is required to change the rotation of the document to zero without altering its appearance. In addition, it makes sure the media box has its origin at (0,0), changing other boxes to compensate. This is important because some operations in CPDF (such as `scale-to-fit`), and in other PDF-processing programs, work properly only when the origin is (0, 0).

```
cpdf -upright in.pdf -o out.pdf
```

Make pages upright.

### 3.5 Flipping Pages

The `-hflip` and `-vflip` operations flip the contents of the chosen pages horizontally or vertically. No account is taken of the current page rotation when considering what “horizontally” and “vertically” mean, so you may like to use `-upright` (see above) first.

```
cpdf -hflip in.pdf even -o out.pdf
```

Flip the even pages in `in.pdf` horizontally.

```
cpdf -vflip in.pdf -o out.pdf
```

Flip all the pages in `in.pdf` vertically.

### 3.6 Boxes and Cropping

All PDF files contain a *media box* for each page, giving the dimensions of the paper. To change these dimensions (without altering the page contents in any way), use the `-mediabox` operation.

```
cpdf -mediabox "0pt 0pt 500pt 500pt" in.pdf -o out.pdf
```

Set the media box to 500 points square.

The four numbers are minimum x, minimum y, width, height. x coordinates increase to the right, y coordinates increase upwards. PDF file can also optionally contain a *crop box* for each page, defining to what extent the page is cropped before being displayed or printed. A crop box can be set, changed and removed, without affecting the underlying media box. To set or change the crop box use `-cropbox`. To remove any existing crop box, use `-remove-cropbox`.

```
cpdf -cropbox "0pt 0pt 200mm 200mm" in.pdf -o out.pdf
```

Crop pages to the bottom left 200-millimeter square of the page.

```
cpdf -remove-cropbox in.pdf -o out.pdf
```

Remove cropping.

Note that the crop box is only obeyed in some viewers. Similar operations are available for the bleed, art, and trim boxes (`-art`, `-remove-bleed` etc.)

```
cpdf -frombox <boxname> -tobox <boxname> [-mediabox-if-missing]
in.pdf [<range>] -o out.pdf
```

Copy the contents of one box to another.

This operation copies the contents of one box (Media box, Crop box, Trim box etc.) to another. If `-mediabox-if-missing` is added, the media box will be substituted when the 'from' box is not set for a given page. For example

```
cpdf -frombox /TrimBox -tobox /CropBox in.pdf -o out.pdf
```

copies the Trim Box of each page to the Crop Box of each page. The possible boxes are `/MediaBox`, `/CropBox`, `/BleedBox`, `/TrimBox`, `/ArtBox`.

A hard box (one which clips its contents by inserting a clipping rectangle) may be created with the `-hard-box` operation:

```
cpdf -hard-box /TrimBox in.pdf -o out.pdf
```

This means the resultant file may be used as a stamp without contents outside the given box reappearing. The `-mediabox-if-missing` option may also be used here.

### 3.7 Showing Boxes and Printer's Marks

The `-show-boxes` operation displays the boxes present on each page as method of debugging. Since boxes may be coincident, they are shown in differing colours and dash patterns so they may be identified even where they overlap. The colours are:

Media box	Red
Crop box	Green
Art box	Blue
Trim box	Orange
Bleed box	Pink

The `-trim-marks` operation adds trim marks to a PDF file. The trim box must be present.



## .NET Interface

### CHAPTER 3. Pages

`Cpdf.scalePages(Cpdf.Pdf, List{Int32}, Double, Double)`

Scales the page dimensions and content by the given scale, about (0, 0). Other boxes (crop etc. are altered as appropriate)

`Cpdf.scaleToFit(Cpdf.Pdf, List{Int32}, Double, Double, Double)`

Scales the content to fit new page dimensions (width x height) multiplied by scale (typically 1.0). Other boxes (crop etc. are altered as appropriate)

`Cpdf.Papersize`

Built-in paper sizes

`Cpdf.Papersize.A0portrait`

A0 Portrait paper

`Cpdf.Papersize.A1portrait`

A1 Portrait paper

`Cpdf.Papersize.A2portrait`

A2 Portrait paper

`Cpdf.Papersize.A3portrait`

A3 Portrait paper

`Cpdf.Papersize.A4portrait`

A4 Portrait paper

`Cpdf.Papersize.A5portrait`

A5 Portrait paper

`Cpdf.Papersize.A0landscape`

A0 Landscape paper

`Cpdf.Papersize.A1landscape`

A1 Landscape paper

`Cpdf.Papersize.A2landscape`

A2 Landscape paper

`Cpdf.Papersize.A3landscape`

A3 Landscape paper

`Cpdf.Papersize.A4landscape`

A4 Landscape paper

`Cpdf.Papersize.A5landscape`

A5 Landscape paper

`Cpdf.Papersize.Usletterportrait`

US Letter Portrait paper

`Cpdf.Papersize.Usletterlandscape`

US Letter Landscape paper

`Cpdf.Papersize.Uslegalportrait`

US Legal Portrait paper

`Cpdf.Papersize.Uslegallandscape`

US Legal Landscape paper

`Cpdf.scaleToFitPaper(Cpdf.Pdf, List{Int32}, Cpdf.Papersize, Double)`

Scales the page content to fit the given page size, possibly multiplied by scale (typically 1.0)

`Cpdf.Anchor`

Position anchors

`Cpdf.Anchor.PosCentre`

Absolute centre

`Cpdf.Anchor.PosLeft`

Absolute left

`Cpdf.Anchor.PosRight`

Absolute right

`Cpdf.Anchor.Top`

The top centre of the page

`Cpdf.Anchor.TopLeft`

The top left of the page

`Cpdf.Anchor.TopRight`

The top right of the page

`Cpdf.Anchor.Left`

The left hand side of the page, halfway down

`Cpdf.Anchor.BottomLeft`

The bottom left of the page

`Cpdf.Anchor.Bottom`

The bottom middle of the page

`Cpdf.Anchor.BottomRight`

The bottom right of the page

`Cpdf.Anchor.Right`

The right hand side of the page, halfway down

`Cpdf.Anchor.Diagonal`

Diagonal, bottom left to top right

`Cpdf.Anchor.ReverseDiagonal`

Diagonal, top left to bottom right

`Cpdf.Position`

`Cpdf.Position.anchor`

Position anchor

`Cpdf.Position.coord1`

Parameter one

`Cpdf.Position.coord2`

Parameter two

`Cpdf.Position.#ctor(Cpdf.Anchor)`

Build a position with zero parameters

`Cpdf.Position.#ctor(Cpdf.Anchor, Double)`

Build a position with one parameter

`Cpdf.Position.#ctor(Cpdf.Anchor, Double, Double)`

Build a position with two parameters

`Cpdf.scaleContents(Cpdf.Pdf, List{Int32}, Cpdf.Position, Double)`

Scales the contents of the pages in the range about the point given by the position, by the scale given.

`Cpdf.shiftContents(Cpdf.Pdf, List{Int32}, Double, Double)`

Shifts the content of the pages in the range.

`Cpdf.rotate(Cpdf.Pdf, List{Int32}, Int32)`

Changes the viewing rotation to an absolute value. Appropriate rotations are 0, 90, 180, 270.

`Cpdf.rotateBy(Cpdf.Pdf, List{Int32}, Int32)`

Rotates the content about the centre of the page by the given number of degrees, in a clockwise direction.

`Cpdf.rotateContents(Cpdf.Pdf, List{Int32}, Double)`

Rotates the content about the centre of the page by the given number of degrees, in a clockwise direction.

`Cpdf.upright(Cpdf.Pdf, List{Int32})`

Changes the viewing rotation of the pages in the range, counter-rotating the dimensions and content such that there is no visual change.

`Cpdf.hFlip(Cpdf.Pdf, List{Int32})`

Flips horizontally the pages in the range.

`Cpdf.vFlip(Cpdf.Pdf, List{Int32})`

Flips vertically the pages in the range.

```
Cpdf.crop(Cpdf.Pdf, List{Int32}, Double, Double, Double, Double)
```

Crops a page, replacing any existing crop box. The dimensions are in points.

```
Cpdf.removeCrop(Cpdf.Pdf, List{Int32})
```

Removes any crop box from pages in the range.

```
Cpdf.removeTrim(Cpdf.Pdf, List{Int32})
```

Removes any trim box from pages in the range.

```
Cpdf.removeArt(Cpdf.Pdf, List{Int32})
```

Removes any art box from pages in the range.

```
Cpdf.removeBleed(Cpdf.Pdf, List{Int32})
```

Removes any bleed box from pages in the range.

```
Cpdf.trimMarks(Cpdf.Pdf, List{Int32})
```

Adds trim marks to the given pages, if the trimbox exists.

```
Cpdf.showBoxes(Cpdf.Pdf, List{Int32})
```

Shows the boxes on the given pages, for debug.

```
Cpdf.hardBox(Cpdf.Pdf, List{Int32}, String)
```

Makes a given box a 'hard box' i.e clips it explicitly.



# Chapter 4

## Encryption and Decryption

```
cpdf -encrypt <method> [-pw=]<owner> [-pw=]<user>  
    [-no-encrypt-metadata] <permissions> in.pdf -o out.pdf  
cpdf -decrypt [-decrypt-force] in.pdf owner=<owner password> -o out.pdf
```

### 4.1 Introduction

PDF files can be encrypted using various types of encryption and attaching various permissions describing what someone can do with a particular document (for instance, printing it or extracting content). There are two types of person:

The **User** can do to the document what is allowed in the permissions.

The **Owner** can do anything, including altering the permissions or removing encryption entirely.

There are five kinds of encryption:

- 40-bit encryption (method 40bit) in Acrobat 3 (PDF 1.1) and above
- 128-bit encryption (method 128bit) in Acrobat 5 (PDF 1.4) and above
- 128-bit AES encryption (method AES) in Acrobat 7 (PDF 1.6) and above
- 256-bit AES encryption (method AES256) in Acrobat 9 (PDF 1.7) – *this is deprecated – do not use for new documents*
- 256-bit AES encryption (method AES256ISO) in PDF 2.0

All encryption supports these kinds of permissions:

-no-edit	Cannot change the document
-no-print	Cannot print the document
-no-copy	Cannot select or copy text or graphics
-no-annot	Cannot add or change form fields or annotations

In addition, 128-bit encryption (Acrobat 5 and above) and AES encryption supports these:

<code>-no-forms</code>	Cannot edit form fields
<code>-no-extract</code>	Cannot extract text or graphics
<code>-no-assemble</code>	Cannot merge files etc.
<code>-no-hq-print</code>	Cannot print high-quality

Add these options to the command line to prevent each operation.

## 4.2 Encrypting a Document

To encrypt a document, the owner and user passwords must be given (here, `fred` and `charles` respectively):

```
cpdf -encrypt 40bit fred charles -no-print in.pdf -o out.pdf
cpdf -encrypt 128bit fred charles -no-extract in.pdf -o out.pdf
cpdf -encrypt AES fred "" -no-edit -no-copy in.pdf -o out.pdf
```

A blank user password is common. In this event, PDF viewers will typically not prompt for a password for when opening the file or for operations allowable with the user password.

```
cpdf -encrypt AES256 fred "" -no-forms in.pdf -o out.pdf
```

In addition, the usual method can be used to give the existing owner password, if the document is already encrypted.

The optional `-pw=` preface may be given where a password might begin with a `-` and thus be confused with a command line option.

When using AES encryption, the option is available to refrain from encrypting the metadata. Add `-no-encrypt-metadata` to the command line.

## 4.3 Decrypting a Document

To decrypt a document, the owner password is provided.

```
cpdf -decrypt in.pdf owner=fred -o out.pdf
```

The user password cannot decrypt a file.

When appropriate passwords are not available, the option `-decrypt-force` may be added to the command line to process the file regardless.



## **.NET Interface**

CHAPTER 4. Encryption

Encryption covered in Chapter 1.



# Chapter 5

## Compression

```
cpdf -decompress in.pdf -o out.pdf
cpdf -compress in.pdf -o out.pdf
cpdf -squeeze in.pdf [-squeeze-log-to <filename>]
    [-squeeze-no-recompress] [-squeeze-no-page-data] -o out.pdf
```

`cpdf` provides basic facilities for decompressing and compressing PDF streams, and for reprocessing the whole file to ‘squeeze’ it.

### 5.1 Decompressing a Document

To decompress the streams in a PDF file, for instance to manually inspect the PDF, use:

```
cpdf -decompress in.pdf -o out.pdf
```

If `cpdf` finds a compression type it can’t cope with, the stream is left compressed. When using `-decompress`, object streams are not compressed. It may be easier for manual inspection to also remove object streams, by adding the `-no-preserve-objstm` option to the command.

### 5.2 Compressing a Document

To compress the streams in a PDF file, use:

```
cpdf -compress in.pdf -o out.pdf
```

`cpdf` compresses any streams which have no compression using the **FlateDecode** method, with the exception of Metadata streams, which are left uncompressed.

## 5.3 Squeezing a Document

To *squeeze* a PDF file, reducing its size by an average of about twenty percent (though sometimes not at all), use:

```
cpdf -squeeze in.pdf -o out.pdf
```

Adding `-squeeze` to the command line when using another operation will *squeeze* the file or files upon output.

The `-squeeze` operation writes some information about the squeezing process to standard output. The squeezing process involves several processes which losslessly attempt to reduce the file size. It is slow, so should not be used without thought.

```
$ ./cpdf -squeeze in.pdf -o out.pdf
Initial file size is 238169 bytes
Beginning squeeze: 123847 objects
Squeezing... Down to 114860 objects
Squeezing... Down to 114842 objects
Squeezing page data
Recompressing document
Final file size is 187200 bytes, 78.60% of original.
```

The `-squeeze-log-to <filename>` option writes the log to the given file instead of to standard output. Log contents is appended to the end of the log file, preserving existing contents.

There are two options which turn off parts of the squeezer. They are `-squeeze-no-recompress` for avoiding the reprocessing of malformed compressed sections, and `-squeeze-no-page-data` for avoiding the reprocessing of malformed page data.

## **.NET Interface**

CHAPTER 5. Compression

`Cpdf.compress(Cpdf.Pdf)`

Compresses any uncompressed streams in the given PDF using the Flate algorithm.

`Cpdf.decompress(Cpdf.Pdf)`

Decompresses any streams in the given PDF, so long as the compression method is supported.

`Cpdf.squeezeInMemory(Cpdf.Pdf)`

Squeezes a pdf in memory.



# Chapter 6

## Bookmarks

```
cpdf -list-bookmarks [-utf8 | -raw] in.pdf
cpdf -list-bookmarks-json in.pdf
cpdf -remove-bookmarks in.pdf -o out.pdf
cpdf -add-bookmarks <bookmark file> in.pdf -o out.pdf
cpdf -add-bookmarks-json <bookmark file> in.pdf -o out.pdf
cpdf -bookmarks-open-to-level <n> in.pdf -o out.pdf
cpdf -table-of-contents [-toc-title] [-toc-no-bookmark]
    [-font <font>] [-font-size <size>] in.pdf -o out.pdf
```

PDF bookmarks (properly called the *document outline*) represent a tree of references to parts of the file, typically displayed at the side of the screen. The user can click on one to move to the specified place. `cpdf` provides facilities to list, add, and remove bookmarks. The format used by the list and add operations is the same, so you can feed the output of one into the other, for instance to copy bookmarks.

### 6.1 List Bookmarks

The `-list-bookmarks` operation prints (to standard output) the bookmarks in a file. The first column gives the level of the tree at which a particular bookmark is. Then the text of the bookmark in quotes. Then the page number which the bookmark points to. Then (optionally) the word "open" if the bookmark should have its children (at the level immediately below) visible when the file is loaded. Then the destination (see below). For example, upon executing

```
cpdf -list-bookmarks doc.pdf
```

the result might be:

```

0 "Part 1" 1 open
1 "Part 1A" 2 "[2 /XYZ 200 400 null]"
1 "Part 1B" 3
0 "Part 2" 4
1 "Part 2a" 5

```

If the page number is 0, it indicates that clicking on that entry doesn't move to a page.

By default, `cpdf` converts unicode to ASCII text, dropping characters outside the ASCII range. To prevent this, and return unicode UTF8 output, add the `-utf8` option to the command. To prevent any processing, use the `-raw` option. See Section 1.17 for more information. A newline in a bookmark is represented as `"\n"`.

By using `-list-bookmarks-json` instead, the bookmarks are formatted as a JSON array, in order, of dictionaries formatted thus:

```

{ "level": 0,
  "text": "1 Basic Usage",
  "page": 17,
  "open": false,
  "target":
    [ { "I": 17 },
      { "N": "/XYZ" },
      { "F": 85.039 },
      { "F": 609.307 },
      null ]
}

```

See chapter 15 for more details of `cpdf`'s JSON formatting. Bookmark text in JSON bookmarks, however, is in UTF8 for ease of use.

### 6.1.1 Destinations

The destination is an extended description of where the bookmark should point to (i.e it can be more detailed than just giving the page). For example, it may point to a section heading halfway down a page. Here are the possibilities:



Format	Description
[ <i>p</i> /XYZ <i>left top zoom</i> ]	Display page number <i>p</i> with ( <i>left, top</i> ) positioned at upper-left of window and magnification of <i>zoom</i> . Writing “null” for any of <i>left, top</i> or <i>zoom</i> specifies no change. A <i>zoom</i> of 0 is the same as “null”.
[ <i>p</i> /Fit]	Display page number <i>p</i> so as to fit fully within the window.
[ <i>p</i> /FitH <i>top</i> ]	Display page number <i>p</i> with vertical coordinate <i>top</i> at the top of the window and the page magnified so its width fits the window. A null value for <i>top</i> implies no change.
[ <i>p</i> /FitV <i>left</i> ]	Display page number <i>p</i> with horizontal coordinate <i>left</i> at the left of the window, and the page magnified so its height fits the window. A null value for <i>left</i> implies no change.
[ <i>p</i> /FitR <i>left bottom right top</i> ]	Display page number <i>p</i> magnified so as to fit entirely within the rectangle specified by the other parameters.
[ <i>p</i> /FitB]	As for /Fit but with the page’s bounding box (see below).
[ <i>p</i> /FitBH <i>top</i> ]	As for /FitH but with the page’s bounding box (see below).
[ <i>p</i> /FitBV <i>left</i> ]	As for /FitV but with the page’s bounding box (see below).

The *bounding box* is the intersection of the page’s crop box and the bounding box of the page contents. Some other kinds of destination may be produced by `-list-bookmarks`. They will be preserved by `-add-bookmarks` and may be edited at your risk.

## 6.2 Remove Bookmarks

The `-remove-bookmarks` operation removes all bookmarks from the file.

```
cpdf -remove-bookmarks in.pdf -o out.pdf
```

## 6.3 Add Bookmarks

The `-add-bookmarks` file adds bookmarks as specified by a *bookmarks file*, a text file in ASCII or UTF8 encoding and in the same format as that produced by the `-list-bookmarks` operation. If there are any bookmarks in the input PDF already, they are discarded. For example, if the file `bookmarks.txt` contains the output from `-list-bookmarks` above, then the command

```
cpdf -add-bookmarks bookmarks.txt in.pdf -o out.pdf
```

adds the bookmarks to the input file, writing to `out.pdf`. An error will be given if the bookmarks file is not in the correct form (in particular, the numbers in the first column which specify the level must form a proper tree with no entry being more than one greater than the last).

Bookmarks in JSON format (see above) may be added with `-add-bookmarks-json`:

```
cpdf -add-bookmarks-json bookmarks.json in.pdf -o out.pdf
```

Remember that strings in JSON bookmark files are in UTF8, rather than as native PDF strings.

## 6.4 Opening bookmarks

As an alternative to extracting a bookmark file and manipulating the open-status of bookmarks, mass manipulation may be achieved by the following operation:

```
cpdf -bookmarks-open-to-level <level> in.pdf -o out.pdf
```

A level of 0 will close all bookmarks, level 1 will open just the top level, closing all others etc. To open all of them, pick a sufficiently large level.

## 6.5 Making a Table of Contents

Cpdf can automatically generate a table of contents from existing bookmarks, adding it to the beginning of the document.

```
cpdf -table-of-contents in.pdf -o out.pdf
```

The page(s) added will have the same dimensions, media and crop boxes as the first page of the original file. The default title is "Table of Contents", though this may be changed:

```
cpdf -table-of-contents -toc-title "Contents" in.pdf -o out.pdf
```

An empty title removes the title. The sequence `\n` may be used to split the title into lines. The default font is 12pt Times Roman (and 24pt for the title). The base font and size may be changed with `-font` and `-font-size` (see chapter 8 for full details):

```
cpdf -table-of-contents -font "Courier-Bold" -font-size 8
in.pdf -o out.pdf
```

By default, an entry for the new table of contents will be added to the document's bookmarks. To suppress this behaviour, add `-toc-no-bookmark`:

```
cpdf -table-of-contents -toc-no-bookmark in.pdf -o out.pdf
```

# .NET Interface

## CHAPTER 6. Bookmarks

`Cpdf.startGetBookmarkInfo(Cpdf.Pdf)`

Starts the bookmark retrieval process for a given PDF.

`Cpdf.numberBookmarks`

Gets the number of bookmarks for the PDF given to `startGetBookmarkInfo`.

`Cpdf.getBookmarkLevel(Int32)`

Gets the bookmark level for the given bookmark (0...(n - 1)).

`Cpdf.getBookmarkPage(Cpdf.Pdf, Int32)`

Gets the bookmark target page for the given PDF (which must be the same as the PDF passed to `startSetBookmarkInfo`) and bookmark (0...(n - 1)).

`Cpdf.getBookmarkText(Int32)`

Returns the text of bookmark (0...(n - 1)).

`Cpdf.getBookmarkOpenStatus(Int32)`

True if the bookmark is open.

`Cpdf.endGetBookmarkInfo`

Ends the bookmark retrieval process, cleaning up.

`Cpdf.startSetBookmarkInfo(Int32)`

Starts the bookmark setting process for n bookmarks.

`Cpdf.setBookmarkLevel(Int32, Int32)`

Set bookmark level for the given bookmark (0...(n - 1)).

`Cpdf.setBookmarkPage(Cpdf.Pdf, Int32, Int32)`

Sets the bookmark target page for the given PDF (which must be the same as the PDF to be passed to `endSetBookmarkInfo`) and bookmark (0...(n - 1)).

`Cpdf.setBookmarkOpenStatus(Int32, Boolean)`

Sets the open status of bookmark (0...(n - 1)).

`Cpdf.setBookmarkText(Int32, String)`

Sets the text of bookmark (0...(n - 1)).

`Cpdf.endSetBookmarkInfo(Cpdf.Pdf)`

Ends the bookmark setting process, writing the bookmarks to the given PDF.

`Cpdf.getBookmarksJSON(Cpdf.Pdf)`

Returns the bookmark data in JSON format.

`Cpdf.setBookmarksJSON(Cpdf.Pdf, Byte[])`

Sets the bookmarks from JSON bookmark data.

`Cpdf.tableOfContents(Cpdf.Pdf, Cpdf.Font, Double, String, Boolean)`

Typesets a table of contents from existing bookmarks and prepends it to the document. If bookmark is set, the table of contents gets its own bookmark.



# Chapter 7

## Presentations

```
cpdf -presentation in.pdf [<range>] -o out.pdf
      [-trans <transition-name>] [-duration <float>]
      [-vertical] [-outward] [-direction <int>]
      [-effect-duration <float>]
```

The PDF file format, starting at Version 1.1, provides for simple slide-show presentations in the manner of Microsoft Powerpoint. These can be played in Acrobat and possibly other PDF viewers, typically started by entering full-screen mode. The `-presentation` operation allows such a presentation to be built from any PDF file.

The `-trans` option chooses the transition style. When a page range is used, it is the transition *from* each page named which is altered. The following transition styles are available:

**Split** Two lines sweep across the screen, revealing the new page. By default the lines are horizontal. Vertical lines are selected by using the `-vertical` option.

**Blinds** Multiple lines sweep across the screen, revealing the new page. By default the lines are horizontal. Vertical lines are selected by using the `-vertical` option.

**Box** A rectangular box sweeps inward from the edges of the page. Use `-outward` to make it sweep from the center to the edges.

**Wipe** A single line sweeps across the screen from one edge to the other in a direction specified by the `-direction` option.

**Dissolve** The old page dissolves gradually to reveal the new one.

**Glitter** The same as **Dissolve** but the effect sweeps across the page in the direction specified by the `-direction` option.

To remove a transition style currently applied to the selected pages, omit the `-trans` option.

The `-effect-duration` option specifies the length of time in seconds for the transition itself. The default value is one second.

The `-duration` option specifies the maximum time in seconds that the page is displayed before the presentation automatically advances. The default, in the absence of the `-duration` option, is for no automatic advancement.

The `-direction` option (for **Wipe** and **Glitter** styles only) specifies the direction of the effect. The following values are valid:

0 Left to right

90 Bottom to top (**Wipe** only)

180 Right to left (**Wipe** only)

270 Top to bottom

315 Top-left to bottom-right (**Glitter** only)

For example:

```
cpdf -presentation in.pdf 2-end -trans Split -duration 10 -o out.pdf
```

The **Split** style, with vertical lines, and each slide staying ten seconds unless manually advanced. The first page (being a title) does not move on automatically, and has no transition effect.

To use different options on different page ranges, run `cpdf` multiple times on the file using a different page range each time.



## **.NET Interface**

CHAPTER 7. Presentations

Not included in the library version of cpdf.



## Chapter 8

# Watermarks and Stamps

```
cpdf -stamp-on source.pdf
    [-scale-stamp-to-fit] [<positioning command>] [-relative-to-cropbox]
    in.pdf [<range>] [-fast] -o out.pdf

cpdf -stamp-under source.pdf
    [-scale-stamp-to-fit] [<positioning command>] [-relative-to-cropbox]
    in.pdf [<range>] [-fast] -o out.pdf

cpdf -combine-pages over.pdf under.pdf
    [-fast] [-prerotate] [-no-warn-rotate] -o out.pdf

cpdf ([-add-text <text-format> | -add-rectangle <size>])
    [-font <fontname>] [-font-size <size-in-points>]
    [-color <color>] [-line-spacing <number>]
    [-outline] [-linewidth <number>]
    [-underneath] [-relative-to-cropbox]
    [-prerotate] [-no-warn-rotate]
    [-bates <number>] [-bates-at-range <number>]
    [-bates-pad-to <number>] [-opacity <number>]
    [-midline] [-topline]
    [-fast]
    in.pdf [<range>] -o out.pdf
```

See also positioning commands below.

```
cpdf -remove-text in.pdf [<range>] -o out.pdf
cpdf -prepend-content <content> in.pdf [<range>] -o out.pdf
cpdf -postpend-content <content> in.pdf [<range>] -o out.pdf
cpdf -stamp-as-xobject stamp.pdf in.pdf [<range>] -o out.pdf
```

NB: See discussion of `-fast` in Section 1.13.

## 8.1 Add a Watermark or Logo

The `-stamp-on` and `-stamp-under` operations stamp the first page of a source PDF onto or under each page in the given range of the input file. For example,

```
cpdf -stamp-on logo.pdf in.pdf odd -o out.pdf
```

stamps the file `logo.pdf` onto the odd pages of `in.pdf`, writing to `out.pdf`. A watermark should go underneath each page:

```
cpdf -stamp-under topsecret.pdf in.pdf -o out.pdf
```

The position commands in Section 8.2.4 can be used to locate the stamp more precisely (they are calculated relative to the crop box of the stamp). Or, preprocess the stamp with `-shift` first.

The `-scale-stamp-to-fit` option can be added to scale the stamp to fit the page before applying it. The use of positioning commands together with `-scale-stamp-to-fit` is not recommended.

The `-combine-pages` operation takes two PDF files and stamps each page of one over each page of the other. The length of the output is the same as the length of the “under” file. For instance:

```
cpdf -combine-pages over.pdf under.pdf -o out.pdf
```

Page attributes (such as the display rotation) are taken from the “under” file. For best results, remove any rotation differences in the two files using `-upright` (see above) first.

The `-relative-to-cropbox` option takes the positioning command to be relative to the crop box of each page rather than the media box.

## 8.2 Stamp Text, Dates and Times.

The `-add-text` operation allows text, dates and times to be stamped over one or more pages of the input at a given position and using a given font, font size and color.

```
cpdf -add-text "Copyright 2014 ACME Corp." in.pdf -o out.pdf
```

The default is black 12pt Times New Roman text in the top left of each page. The text can be placed underneath rather than over the page by adding the `-underneath` option.

Text previously added by `cpdf` may be removed by the `-remove-text` operation.

```
cpdf -remove-text in.pdf -o out.pdf
```

### 8.2.1 Page Numbers

There are various special codes to include the page number in the text:

<code>%Page</code>	Page number in arabic notation (1, 2, 3...)
<code>%PageDiv2</code>	Page number in arabic notation divided by two
<code>%roman</code>	Page number in lower-case roman notation (i, ii, iii...)
<code>%Roman</code>	Page number in upper-case roman notation (I, II, III...)
<code>%EndPage</code>	Last page of document in arabic notation
<code>%Label</code>	The page label of the page
<code>%EndLabel</code>	The page label of the last page
<code>%filename</code>	The full file name of the input document

For example, the format "Page `%Page` of `%EndPage`" might become "Page 5 of 17".

NB: In some circumstances (e.g in batch files) on Microsoft Windows, `%` is a special character, and must be escaped (written as `%%`). Consult your local documentation for details.

### 8.2.2 Date and Time Formats

<code>%a</code>	Abbreviated weekday name (Sun, Mon etc.)
<code>%A</code>	Full weekday name (Sunday, Monday etc.)
<code>%b</code>	Abbreviated month name (Jan, Feb etc.)
<code>%B</code>	Full month name (January, February etc.)
<code>%d</code>	Day of the month (01–31)
<code>%e</code>	Day of the month (1–31)
<code>%H</code>	Hour in 24-hour clock (00–23)
<code>%I</code>	Hour in 12-hour clock (01–12)
<code>%j</code>	Day of the year (001–366)
<code>%m</code>	Month of the year (01–12)
<code>%M</code>	Minute of the hour (00–59)
<code>%p</code>	"a.m" or "p.m"
<code>%S</code>	Second of the minute (00–61)
<code>%T</code>	Same as <code>%H:%M:%S</code>
<code>%u</code>	Weekday (1–7, 1 = Sunday)
<code>%w</code>	Weekday (0–6, 0 = Sunday)
<code>%Y</code>	Year (0000–9999)
<code>%%</code>	The <code>%</code> character.

### 8.2.3 Bates Numbers

Unique page identifiers can be specified by putting `%Bates` in the format. The starting point can be set with the `-bates` option. For example:

```
cpdf -add-text "Page ID: %Bates" -bates 23745 in.pdf -o out.pdf
```

To specify that bates numbering begins at the first page of the range, use `-bates-at-range` instead. This option must be specified after the range is specified. To pad the bates number

up to a given number of leading zeros, use `-bates-pad-to` in addition to either `-bates` or `-bates-at-range`.

### 8.2.4 Position

The position of the text may be specified either in absolute terms:

```
-pos-center "200 200"
```

Position the center of the baseline text at (200pt, 200pt)

```
-pos-left "200 200"
```

Position the left of the baseline of the text at (200pt, 200pt)

```
-pos-right "200 200"
```

Position the right of the baseline of the text at (200pt, 200pt)

Positions relative to certain common points can be set:

<code>-top 10</code>	Center of baseline 10 pts down from the top center
<code>-topleft 10</code>	Left of baseline 10 pts down and in from top left
<code>-topright 10</code>	Right of baseline 10 pts down and left from top right
<code>-left 10</code>	Left of baseline 10 pts in from center left
<code>-bottomleft 10</code>	Left of baseline 10 pts in and up from bottom left
<code>-bottom 10</code>	Center of baseline 10 pts up from bottom center
<code>-bottomright 10</code>	Right of baseline 10 pts up and in from bottom right
<code>-right 10</code>	Right of baseline 10 pts in from the center right
<code>-diagonal</code>	Diagonal, bottom left to top right, centered on page
<code>-reverse-diagonal</code>	Diagonal, top left to bottom right, centered on page
<code>-center</code>	Centered on page

No attempt is made to take account of the page rotation when interpreting the position, so `-prerotate` may be added to the command line if the file contains pages with a non-zero viewing rotation (to silence the rotation warning, add `-no-warn-rotate` instead) This is equivalent to pre-processing the document with `-upright` (see chapter 3).

The `-relative-to-cropbox` modifier can be added to the command line to make these measurements relative to the crop box instead of the media box.

The default position is equivalent to `-topleft 100`.

The `-midline` option may be added to specify that the positioning commands above are to be considered relative to the midline of the text, rather than its baseline. Similarly, the `-topline` option may be used to specify that the position is taken relative to the top of the text.

### 8.2.5 Font and Size

The font may be set with the `-font` option. The 14 Standard PDF fonts are available:

Times-Roman  
 Times-Bold  
 Times-Italic  
 Times-BoldItalic  
 Helvetica  
 Helvetica-Bold  
 Helvetica-Oblique  
 Helvetica-BoldOblique  
 Courier  
 Courier-Bold  
 Courier-Oblique  
 Courier-BoldOblique  
 Symbol  
 ZapfDingbats

For example, page numbers in Times Italic can be achieved by:

```
cpdf -add-text "-%Page-" -font "Times-Italic" in.pdf -o out.pdf
```

See Section 14.3 for how to use other fonts. The font size can be altered with the `-font-size` option, which specifies the size in points:

```
cpdf -add-text "-%Page-" -font-size 36 in.pdf -o out.pdf
```

## 8.2.6 Colors

The `-color` option takes an RGB (3 values), CMYK (4 values), or Grey (1 value) color. Components range between 0 and 1. The following RGB colors are predefined:

Color	R, G, B
white	1, 1, 1
black	0, 0, 0
red	1, 0, 0
green	0, 1, 0
blue	0, 0, 1

```
cpdf -add-text "Hullo" -color "red" in.pdf -o out.pdf
cpdf -add-text "Hullo" -color "0.5 0.5 0.5" in.pdf -o out.pdf
cpdf -add-text "Hullo" -color "0.75" in.pdf -o out.pdf
cpdf -add-text "Hullo" -color "0.5 0.5 0.4 0.9" in.pdf -o out.pdf
```

Partly-transparent text may be specified using the `-opacity` option. Wholly opaque is 1 and wholly transparent is 0. For example:

```
cpdf -add-text "DRAFT" -color "red" -opacity 0.3 -o out.pdf
```

### 8.2.7 Outline Text

The `-outline` option sets outline text. The line width (default 1pt) may be set with the `-linewidth` option. For example, to stamp documents as drafts:

```
cpdf -add-text "DRAFT" -diagonal -outline in.pdf -o out.pdf
```

### 8.2.8 Multi-line Text

The code `\n` can be included in the text string to move to the next line. In this case, the vertical position refers to the baseline of the first line of text (if the position is at the top, top left or top right of the page) or the baseline of the last line of text (if the position is at the bottom, bottom left or bottom right).

```
cpdf -add-text "Specification\n%Page of %EndPage"
          -topright 10 in.pdf -o out.pdf
```

The `-midline` option may be used to make these vertical positions relative to the midline of a line of text rather than the baseline, as usual.

The `-line-spacing` option can be used to increase or decrease the line spacing, where a spacing of 1 is the standard.

```
cpdf -add-text "Specification\n%Page of %EndPage"
          -topright 10 -line-spacing 1.5 in.pdf -o out.pdf
```

Justification of multiple lines is handled by the `-justify-left`, `-justify-right` and `-justify-center` options. The defaults are left justification for positions relative to the left hand side of the page, right justification for those relative to the right, and center justification for positions relative to the center of the page. For example:

```
cpdf -add-text "Long line\nShort" -justify-right
          in.pdf -o out.pdf
```



## 8.2.9 Special Characters

If your command line allows for the inclusion of unicode characters, the input text will be considered as UTF8 by `cpdf`. Special characters which exist in the PDF WinAnsiEncoding Latin 1 code (such as many accented characters) will be reproduced in the PDF. This does not mean, however, that every special character can be reproduced – it must exist in the font. When using a custom font, `cpdf` will attempt to convert from UTF8 to the encoding of that font automatically.

(For compatibility with previous versions of `cpdf`, special characters may be introduced manually with a backslash followed by the three-digit octal code of the character in the PDF WinAnsiEncoding Latin 1 Code. The full table is included in Appendix D of the Adobe PDF Reference Manual, which is available at [https://www.images2.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000\\_2008.pdf](https://www.images2.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf). For example, a German sharp s (ß) may be introduced by `\337.`)

## 8.3 Stamping Graphics

A rectangle may be placed on one or more pages by using the `-add-rectangle <size>` command. Most of the options discussed above for text placement apply in the same way. For example:

```
cpdf -add-rectangle "200 300" -pos-right 30 -color red -outline
in.pdf -o out.pdf
```

This can be used to blank out or highlight part of the document. The following positioning options work as you would expect: `-topleft`, `-top`, `-topright`, `-right`, `-bottomright`, `-bottom`, `-bottomleft`, `-left`, `-center`. When using the option `-pos-left "x y"`, the point (x, y) refers to the bottom-left of the rectangle. When using the option `-pos-right "x y"`, the point (x, y) refers to the bottom-right of the rectangle. When using the option `-pos-center "x y"`, the point (x, y) refers to the center of the rectangle. The options `-diagonal` and `-reverse-diagonal` have no meaning.

## 8.4 Low-level facilities

These two operations add content directly to the beginning or end of the page data for a page. You must understand the PDF page description language to use these.

```
cpdf -prepend-content <content> in.pdf [<range>] -o out.pdf

cpdf -postpend-content <content> in.pdf [<range>] -o out.pdf
```

The `-fast` option may be added (see Chapter 1). The `-stamp-as-xobject` operation puts a file in another as a Form XObject on the given pages. You can then use `-prepend-content` or `-postpend-content` to use it.

```
cpdf -stamp-as-xobject stamp.pdf in.pdf [<range>] -o out.pdf
```

# .NET Interface

## CHAPTER 8. Logos, Watermarks and Stamps

```
Cpdf.stampOn(Cpdf.Pdf, Cpdf.Pdf, List{Int32})
```

Stamps stamp\_pdf on top of all the pages in the document which are in the range. The stamp is placed with its origin at the origin of the target document.

```
Cpdf.stampUnder(Cpdf.Pdf, Cpdf.Pdf, List{Int32})
```

Stamps stamp\_pdf under all the pages in the document which are in the range. The stamp is placed with its origin at the origin of the target document.

```
Cpdf.stampExtended(Cpdf.Pdf, Cpdf.Pdf, List{Int32}, Boolean, Boolean,  
Cpdf.Position, Boolean)
```

A stamping function with extra features. - isover true, pdf goes over pdf2, isover false, pdf goes under pdf2 - scale\_stamp\_to\_fit scales the stamp to fit the page - pos gives the position to put the stamp - relative\_to\_croptbox: if true, pos is relative to croptbox not mediabox.

```
Cpdf.combinePages(Cpdf.Pdf, Cpdf.Pdf)
```

Combines the PDFs page-by-page, putting each page of 'over' over each page of 'under'.

```
Cpdf.Font
```

Standard fonts

```
Cpdf.Font.TimesRoman
```

Times Roman

```
Cpdf.Font.TimesBold
```

Times Bold

```
Cpdf.Font.TimesItalic
```

Times Italic

```
Cpdf.Font.TimesBoldItalic
```

Times Bold Italic

```
Cpdf.Font.Helvetica
```

Helvetica

Cpdf.Font.HelveticaBold

Helvetica Bold

Cpdf.Font.HelveticaOblique

Helvetica Oblique

Cpdf.Font.HelveticaBoldOblique

Helvetica Bold Oblique

Cpdf.Font.Courier

Courier

Cpdf.Font.CourierBold

Courier Bold

Cpdf.Font.CourierOblique

Courier Oblique

Cpdf.Font.CourierBoldOblique

Courier Bold Oblique

Cpdf.Justification

Justifications

Cpdf.Justification.LeftJustify

Left justify

Cpdf.Justification.CentreJustify

Centre justify

Cpdf.Justification.RightJustify

Right justify

Cpdf.addText(Boolean, Cpdf.Pdf, List{Int32}, String, Cpdf.Position, Double, Int32, Cpdf.Font, Double, Double, Double, Double, Boolean, Boolean, Boolean, Double, Cpdf.Justification, Boolean, Boolean, String, Double, Boolean)

Adds text to the pages in the given range.

`Cpdf.addTextSimple(Cpdf.Pdf, List{Int32}, String, Cpdf.Position, Cpdf.Font, Double)`

Adds text with most parameters default.

`Cpdf.removeText(Cpdf.Pdf, List{Int32})`

Removes any text added by cpdf from the given pages.

`Cpdf.textWidth(Cpdf.Font, String)`

Returns the width of a given string in the given font in thousandths of a point.

`Cpdf.addContent(String, Boolean, Cpdf.Pdf, List{Int32})`

Adds page content before (if true) or after (if false) the existing content to pages in the given range in the given PDF.

`Cpdf.stampAsXObject(Cpdf.Pdf, List{Int32}, Cpdf.Pdf)`

Stamps `stamp_pdf` onto the pages in the given range in pdf as a shared Form XObject. The name of the newly-created XObject is returned.



## Chapter 9

# Multipage Facilities

```
cpdf -pad-before in.pdf [<range>] [-pad-with pad.pdf] -o out.pdf
cpdf -pad-after in.pdf [<range>] [-pad-with pad.pdf] -o out.pdf
cpdf -pad-every [<integer>] in.pdf [-pad-with pad.pdf] -o out.pdf
cpdf -pad-multiple [<integer>] in.pdf -o out.pdf
cpdf -pad-multiple-before [<integer>] in.pdf -o out.pdf
cpdf [-impose <pagesize> | impose-xy <x y>]
    [-impose-columns] [-impose-rtl] [-impose-btt]
    [-impose-margin <margin>] [-impose-spacing <spacing>]
    [-impose-linewidth <width>] [-fast]
    in.pdf -o out.pdf
cpdf -twoup-stack [-fast] in.pdf -o out.pdf
cpdf -twoup [-fast] in.pdf -o out.pdf
```

### 9.1 Inserting Blank Pages

Sometimes, for instance to get a printing arrangement right, it's useful to be able to insert blank pages into a PDF file. `cpdf` can add blank pages before a given page or pages, or after. The pages in question are specified by a range in the usual way:

```
cpdf -pad-before in.pdf 1 -o out.pdf
```

Add a blank page before page 1 (i.e. at the beginning of the document.)

```
cpdf -pad-after in.pdf 2,16,38,84,121,147 -o out.pdf
```

Add a blank page after pages 2, 16, 38, 84, 121 and 147 (for instance, to add a clean page between chapters of a document.)

The dimensions of the padded page are derived from the boxes (media box, crop box etc.) of the page after or before which the padding is to be applied.

The `-pad-every n` operation places a blank page after every  $n$  pages, excluding any last one. For example...

```
cpdf -pad-every 3 in.pdf -o out.pdf
```

Add a blank page after every three pages

... on a 9 page document adds a blank page after pages 3 and 6.

In all three of these operations, one may specify `-pad-with` providing a (usually one-page) PDF file to be used instead of a blank page. For example, a page saying "This page left intentionally blank".

The `-pad-multiple n` operation adds blank pages so the document has a multiple of  $n$  pages. For example:

```
cpdf -pad-multiple 8 in.pdf -o out.pdf
```

Add blank pages to `in.pdf` so it has a multiple of 8 pages.

The `-pad-multiple-before n` operation adds the padding pages at the beginning of the file instead.

## 9.2 Imposition

Imposition is the act of putting two or more pages of an input document onto each page of the output document. There are two operations provided by `cpdf`:

- the `-impose` operation which, given a page size fits multiple pages into it; and
- the `-impose-xy` operation which, given an  $x$  and  $y$  value, builds an output page which fits  $x$  input pages horizontally and  $y$  input pages vertically.

```
cpdf -impose a0landscape in.pdf -o out.pdf
```

Impose as many pages as will fit on to new A0 landscape pages.

```
cpdf -impose-xy "3 4" in.pdf -o out.pdf
```

Impose 3 across and 4 down on to new pages of 3 times the width and 4 times the height of the input ones.

The  $x$  value for `-impose-xy` may be set to zero to indicate an infinitely-wide page; the  $y$  value to indicate an infinitely-long one.

In both cases, the pages in the input file are assumed to be of the same dimensions.

The following options may be used to modify the output:



- `-impose-columns` Lay the pages out in columns rather than rows.
- `-impose-rtl` Lay the pages out right-to-left.
- `-impose-btt` Lay the pages out bottom-to-top.
- `-impose-margin <margin>` Add a margin around the edge of the page of the given width. When using `-impose-xy` the page size increases; with `-impose` the pages are scaled.
- `-impose-spacing <spacing>` Add spacing between each row and column. When using `-impose-xy` the page size increases; with `-impose` the pages are scaled.
- `-impose-linewidth <width>` Add a border around each input page. With `-impose` the pages are scaled after the border is added, so you must account for this yourself.

To impose with rotated pages, for example to put two A4 portrait pages two-up on an A3 landscape page, rotate them prior to imposition.

Two other ways of putting multiple pages on a single page remain from earlier versions of `cpdf` which lacked a general imposition operation. The `-twoup-stack` operation puts two logical pages on each physical page, rotating them 90 degrees to do so. The new mediabox is thus larger. The `-twoup` operation does the same, but scales the new sides down so that the media box is unchanged.

```
cpdf -two-up in.pdf -o out.pdf
```

Impose a document two-up, keeping the existing page size.

```
cpdf -two-up-stack in.pdf -o out.pdf
```

Impose a document two-up on a larger page by rotation.

NB: For all imposition options, see also discussion of `-fast` in Section 1.13.

## .NET Interface

### CHAPTER 9. Multipage facilities

```
Cpdf.impose(Cpdf.Pdf, Double, Double, Boolean, Boolean, Boolean, Boolean,
Boolean, Double, Double, Double)
```

Imposes a PDF. There are two modes: imposing  $x * y$ , or imposing to fit a page of size  $x * y$ . This is controlled by `fit`. `Columns` imposes by columns rather than rows. `rtl` is right-to-left, `btt` bottom-to-top. `Center` is unused for now. `Margin` is the margin around the output, spacing the spacing between imposed inputs.

```
Cpdf.twoUp(Cpdf.Pdf)
```

Imposes a document two up. `twoUpStack` does so by doubling the page size, to fit two pages on one.

```
Cpdf.twoUpStack(Cpdf.Pdf)
```

Impose a document two up. `twoUpStack` does so by doubling the page size, to fit two pages on one.

```
Cpdf.padBefore(Cpdf.Pdf, List{Int32})
```

Adds a blank page before each page in the given range.

```
Cpdf.padAfter(Cpdf.Pdf, List{Int32})
```

Adds a blank page after each page in the given range.

```
Cpdf.padEvery(Cpdf.Pdf, Int32)
```

Adds a blank page after every `n` pages.

```
Cpdf.padMultiple(Cpdf.Pdf, Int32)
```

Adds pages at the end to pad the file to a multiple of `n` pages in length.

```
Cpdf.padMultipleBefore(Cpdf.Pdf, Int32)
```

Adds pages at the beginning to pad the file to a multiple of `n` pages in length.

# Chapter 10

## Annotations

```
cpdf -list-annotations in.pdf [<range>]
cpdf -list-annotations-json in.pdf [<range>]
cpdf -copy-annotations from.pdf to.pdf [<range>] -o out.pdf
cpdf -remove-annotations in.pdf [<range>] -o out.pdf
```

### 10.1 Listing Annotations

The `-list-annotations` operation prints the textual content of any annotations on the selected pages to standard output. Each annotation is preceded by the page number and followed by a newline. The output of this operation is always UTF8.

```
cpdf -list-annotations in.pdf > annots.txt
```

Print annotations from `in.pdf`, redirecting output to `annots.txt`.

More information can be obtained by listing annotations in JSON format:

```
cpdf -list-annotations-json in.pdf > annots.json
```

Print annotations from `in.pdf` in JSON format, redirecting output to `annots.json`.

This produces an array of (page number, annotation) pairs giving the PDF structure of each annotation. Destination pages for page links will have page numbers in place of internal PDF page links, and certain indirect objects are made direct but the content is otherwise unaltered. Here is an example entry for an annotation on page 10:

```
[
  10,
```

```
{ "/H": { "N": "/I" },
  "/Border": [ { "I": 0 }, { "I": 0 }, { "I": 0 } ],
  "/Rect": [
    { "F": 89.88023 }, { "F": 409.98401 }, { "F": 323.90561 }, {
      "F": 423.32059 } ],
  "/Subtype": { "N": "/Link" },
  "/Type": { "N": "/Annot" },
  "/A": {
    "/S": { "N": "/URI" },
    "/URI": "http://www.google.com/" },
  "/StructParent": { "I": 10 } } ]
```

A future version of `cpdf` will allow these JSON annotations to be edited and re-loaded into a PDF file.

## 10.2 Copying Annotations

The `-copy-annotations` operation copies the annotations in the given page range from one file (the file specified immediately after the option) to another pre-existing PDF. The range is specified after this pre-existing PDF. The result is then written an output file, specified in the usual way.

```
cpdf -copy-annotations from.pdf to.pdf 1-10 -o result.pdf
```

Copy annotations from the first ten pages of `from.pdf` onto the PDF file `to.pdf`, writing the result to `results.pdf`.

## 10.3 Removing Annotations

The `-remove-annotations` operation removes all annotations from the given page range.

```
cpdf -remove-annotations in.pdf 1 -o out.pdf
```

Remove annotations from the first page of a file only.

## **.NET Interface**

CHAPTER 10. Annotations

`Cpdf.annotationsJSON(Cpdf.Pdf)`

Returns the annotations from a PDF in JSON format



## Chapter 11

# Document Information and Metadata

```
cpdf -info [-raw | -utf8] in.pdf
cpdf -page-info in.pdf
cpdf -pages in.pdf
cpdf -set-title <title of document>
    [-also-set-xmp] [-just-set-xmp] [-raw] in.pdf -o out.pdf
(Also -set-author etc. See Section 11.2.)
cpdf -set-page-layout <layout> in.pdf -o out.pdf
cpdf -set-page-mode <mode> in.pdf -o out.pdf
cpdf -hide-toolbar <true | false> in.pdf -o out.pdf
    -hide-menubar
    -hide-window-ui
    -fit-window
    -center-window
    -display-doc-title
cpdf -open-at-page <page number> in.pdf -o out.pdf
cpdf -open-at-page-fit <page number> in.pdf -o out.pdf
cpdf -set-metadata <metadata-file> in.pdf -o out.pdf
cpdf -remove-metadata in.pdf -o out.pdf
cpdf -print-metadata in.pdf
cpdf -create-metadata in.pdf -o out.pdf
cpdf -set-metadata-date <date> in.pdf -o out.pdf
cpdf -add-page-labels in.pdf -o out.pdf
    [-label-style <style>] [-label-prefix <string>]
    [-label-startval <integer>] [-labels-progress]
```

```
cpdf -remove-page-labels in.pdf -o out.pdf
cpdf -print-page-labels in.pdf
```

## 11.1 Reading Document Information

The `-info` operation prints entries from the document information dictionary, and from any XMP metadata to standard output.

```
$cpdf -info pdf_reference.pdf
Encryption: 40bit
Linearized: true
Permissions: No edit
Version: 1.6
Pages: 1310
Title: PDF Reference, version 1.7
Author: Adobe Systems Incorporated
Subject: Adobe Portable Document Format (PDF)
Keywords:
Creator: FrameMaker 7.2
Producer: Acrobat Distiller 7.0.5 (Windows)
Created: D:20061017081020Z
Modified: D:20061118211043-02'30'
XMP pdf:Producer: Adobe PDF library 7.77
XMP xmp:CreateDate: 2006-12-21T18:19:09+01:00
XMP xmp:CreatorTool: Adobe Illustrator CS2
XMP xmp:MetadataDate: 2006-12-21T18:19:09Z
XMP xmp:ModifyDate: 2006-12-21T18:19:09Z
XMP dc:title: AI6
```

The details of the format for creation and modification dates can be found in Appendix A.

By default, `cpdf` strips to ASCII, discarding character codes in excess of 127. In order to preserve the original unicode, add the `-utf8` option. To disable all postprocessing of the string, add `-raw`. See Section 1.17 for more information.

The `-page-info` operation prints the page label, media box and other boxes page-by-page to standard output, for all pages in the current range.

```
$cpdf -page-info 14psfonts.pdf
Page 1:
Label: i
MediaBox: 0.000000 0.000000 600.000000 450.000000
CropBox: 200.000000 200.000000 500.000000 500.000000
BleedBox:
```



```
TrimBox:
ArtBox:
Rotation: 0
```

Note that the format for boxes is minimum x, minimum y, maximum x, maximum y.

The `-pages` operation prints the number of pages in the file.

```
cpdf -pages Archos.pdf
8
```

## 11.2 Setting Document Information

The *document information dictionary* in a PDF file specifies various pieces of information about a PDF. These can be consulted in a PDF viewer (for instance, Acrobat).

Here is a summary of the commands for setting entries in the document information dictionary:

Information	Example command-line fragment
Title	<code>cpdf -set-title "Discourses"</code>
Author	<code>cpdf -set-author "Joe Smith"</code>
Subject	<code>cpdf -set-subject "Behavior"</code>
Keywords	<code>cpdf -set-keywords "Ape Primate"</code>
Creator	<code>cpdf -set-creator "Original Program"</code>
Producer	<code>cpdf -set-producer "Distilling Program"</code>
Creation Date	<code>cpdf -set-create "D:19970915110347-08'00' "</code>
Modification Date	<code>cpdf -set-modify "D:19970915110347-08'00' "</code>
Mark as Trapped	<code>cpdf -set-trapped</code>
Mark as Untrapped	<code>cpdf -set-untrapped</code>

(The details of the format for creation and modification dates can be found in Appendix A. Using the date "now" uses the time and date at which the command is executed. Note also that `-producer` and `-creator` may be used to set the producer and/or the creator when writing any file, separate from the operations described in this chapter.)

For example, to set the title, the full command line would be

```
cpdf -set-title "A Night in London" in.pdf -o out.pdf
```

The text string is considered to be in UTF8 format, unless the `-raw` option is added—in which case, it is unprocessed, save for the replacement of any octal escape sequence such as `\017`, which is replaced by a character of its value (here, 15).

To set also any field in the XMP metadata, add `-also-set-xmp`. The field must exist already. To set only the field (not the document information dictionary), add `-just-set-xmp` instead.

## 11.3 XMP Metadata

PDF files can contain a piece of arbitrary metadata, often in XMP format. This is typically stored in an uncompressed stream, so that other applications can read it without having to decode the whole PDF. To set the metadata:

```
cpdf -set-metadata data.xml in.pdf -o out.pdf
```

To remove any metadata:

```
cpdf -remove-metadata in.pdf -o out.pdf
```

To print the current metadata to standard output:

```
cpdf -print-metadata in.pdf
```

To create XMP metadata from scratch, using any information in the Document Information Dictionary (old-style metadata):

```
cpdf -create-metadata in.pdf -o out.pdf
```

To set the XMP metadata date field, use:

```
cpdf -set-metadata-date <date> in.pdf -o out.pdf
```

The date format is defined in Appendix A.2. Using the date "now" uses the time and date at which the command is executed.

## 11.4 Upon Opening a Document

### 11.4.1 Page Layout

The `-set-page-layout` operation specifies the page layout to be used when a document is opened in, for instance, Acrobat. The possible (case-sensitive) values are:

SinglePage	Display one page at a time
OneColumn	Display the pages in one column
TwoColumnLeft	Display the pages in two columns, odd numbered pages on the left
TwoColumnRight	Display the pages in two columns, even numbered pages on the left
TwoPageLeft	(PDF 1.5 and above) Display the pages two at a time, odd numbered pages on the left
TwoPageRight	(PDF 1.5 and above) Display the pages two at a time, even numbered pages on the left

For instance:

```
cpdf -set-page-layout TwoColumnRight in.pdf -o out.pdf
```

NB: If the file has a valid `/OpenAction` setting, which tells the PDF reader to open at a certain page or position on a page, this will override the page layout option. To prevent this, use the `-remove-dict-entry` functionality from Section 18.9:

```
cpdf -remove-dict-entry /OpenAction in.pdf -o out.pdf
```

## 11.4.2 Page Mode

The *page mode* in a PDF file defines how a viewer should display the document when first opened. The possible (case-sensitive) values are:

UseNone	Neither document outline nor thumbnail images visible
UseOutlines	Document outline (bookmarks) visible
UseThumbs	Thumbnail images visible
FullScreen	Full-screen mode (no menu bar, window controls, or anything but the document visible)
UseOC	(PDF 1.5 and above) Optional content group panel visible
UseAttachments	(PDF 1.5 and above) Attachments panel visible

For instance:

```
cpdf -set-page-mode FullScreen in.pdf -o out.pdf
```

### 11.4.3 Display Options

<code>-hide-toolbar</code>	Hide the viewer's toolbar
<code>-hide-menubar</code>	Document outline (bookmarks) visible
<code>-hide-window-ui</code>	Hide the viewer's scroll bars
<code>-fit-window</code>	Resize the document's windows to fit size of first page
<code>-center-window</code>	Position the document window in the center of the screen
<code>-display-doc-title</code>	Display the document title instead of the file name in the title bar

For instance:

```
cpdf -hide-toolbar true in.pdf -o out.pdf
```

The page a PDF file opens at can be set using `-open-at-page`:

```
cpdf -open-at-page 15 in.pdf -o out.pdf
```

To have that page scaled to fit the window in the viewer, use `-open-at-page-fit` instead:

```
cpdf -open-at-page-fit end in.pdf -o out.pdf
```

(Here, we used `end` to open at the last page. Any page specification describing a single page is ok here.)

## 11.5 Page Labels

It is possible to add *page labels* to a document. These are not the printed on the page, but may be displayed alongside thumbnails or in print dialogue boxes by PDF readers. We use `-add-page-labels` to do this, by default with decimal arabic numbers (1,2,3...). We can add `-label-style` to choose what type of labels to add from these kinds:

DecimalArabic	1, 2, 3, 4, 5...
LowercaseRoman	i, ii, iii, iv, v...
UppercaseRoman	I, II, III, IV, V...
LowercaseLetters	a, b, c, ..., z, aa, bb...
UppercaseLetters	A, B, C, ..., Z, AA, BB...
NoLabelPrefixOnly	No number, but a prefix will be used if defined.

We can use `-label-prefix` to add a textual prefix to each label. Consider a file with twenty pages and no current page labels (a PDF reader will assume 1,2,3... if there are none). We will add the following page labels:

i, ii, iii, iv, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A-0, A-1, A-2, A-3, A-4, A-5

Here are the commands, in order:

```
cpdf -add-page-labels in.pdf 1-4 -label-style LowercaseRoman
-o out.pdf

cpdf -add-page-labels out.pdf 5-14 -o out.pdf

cpdf -add-page-labels out.pdf 15-20 -label-prefix "A-"
-label-startval 0 -o out.pdf
```

By default the labels begin at page number 1 for each range. To override this, we can use `-label-startval` (we used 0 in the final command), where we want the numbers to begin at zero rather than one. The option `-labels-progress` can be added to make sure the start value progresses between sub-ranges when the page range specified is disjoint, e.g 1-9, 30-40 or odd.

Page labels may be removed altogether by using `-remove-page-labels` command. To print the page labels from an existing file, use `-print-page-labels`. For example:

```
$ cpdf -print-page-labels cpdfmanual.pdf
labelstyle: LowercaseRoman
labelprefix: None
startpage: 1
startvalue: 1
labelstyle: DecimalArabic
labelprefix: None
startpage: 9
startvalue: 1
```

# .NET Interface

## CHAPTER 11. Document Information and Metadata

`Cpdf.isLinearized(String)`

Finds out if a document is linearized as quickly as possible without loading it.

`Cpdf.getVersion(Cpdf.Pdf)`

Returns the minor version number of a document.

`Cpdf.getMajorVersion(Cpdf.Pdf)`

Returns the minor version number of a document.

`Cpdf.getTitle(Cpdf.Pdf)`

Returns the title of a document.

`Cpdf.getAuthor(Cpdf.Pdf)`

Returns the author of a document.

`Cpdf.getSubject(Cpdf.Pdf)`

Returns the subject of a document.

`Cpdf.getKeywords(Cpdf.Pdf)`

Returns the keywords of a document.

`Cpdf.getCreator(Cpdf.Pdf)`

Returns the creator of a document.

`Cpdf.getProducer(Cpdf.Pdf)`

Returns the producer of a document.

`Cpdf.getCreationDate(Cpdf.Pdf)`

Returns the creation date of a document.

`Cpdf.getModificationDate(Cpdf.Pdf)`

Returns the modification date of a document.

`Cpdf.getTitleXMP(Cpdf.Pdf)`

Returns the XMP title of a document.

```
Cpdf.getAuthorXMP(Cpdf.Pdf)
```

Returns the XMP author of a document.

```
Cpdf.getSubjectXMP(Cpdf.Pdf)
```

Returns the XMP subject of a document.

```
Cpdf.getKeywordsXMP(Cpdf.Pdf)
```

Returns the XMP keywords of a document.

```
Cpdf.getCreatorXMP(Cpdf.Pdf)
```

Returns the XMP creator of a document.

```
Cpdf.getProducerXMP(Cpdf.Pdf)
```

Returns the XMP producer of a document.

```
Cpdf.getCreationDateXMP(Cpdf.Pdf)
```

Returns the XMP creation date of a document.

```
Cpdf.getModificationDateXMP(Cpdf.Pdf)
```

Returns the XMP modification date of a document.

```
Cpdf.setTitle(Cpdf.Pdf, String)
```

Sets the title of a document.

```
Cpdf.setAuthor(Cpdf.Pdf, String)
```

Sets the author of a document.

```
Cpdf.setSubject(Cpdf.Pdf, String)
```

Sets the subject of a document.

```
Cpdf.setKeywords(Cpdf.Pdf, String)
```

Sets the keywords of a document.

```
Cpdf.setCreator(Cpdf.Pdf, String)
```

Sets the creator of a document.

`Cpdf.setProducer(Cpdf.Pdf, String)`

Sets the producer of a document.

`Cpdf.setCreationDate(Cpdf.Pdf, String)`

Sets the creation date of a document.

`Cpdf.setModificationDate(Cpdf.Pdf, String)`

Sets the modification date of a document.

`Cpdf.setTitleXMP(Cpdf.Pdf, String)`

Sets the XMP title of a document.

`Cpdf.setAuthorXMP(Cpdf.Pdf, String)`

Sets the XMP author of a document.

`Cpdf.setSubjectXMP(Cpdf.Pdf, String)`

Sets the XMP subject of a document.

`Cpdf.setKeywordsXMP(Cpdf.Pdf, String)`

Sets the XMP keywords of a document.

`Cpdf.setCreatorXMP(Cpdf.Pdf, String)`

Sets the XMP creator of a document.

`Cpdf.setProducerXMP(Cpdf.Pdf, String)`

Sets the XMP producer of a document.

`Cpdf.setCreationDateXMP(Cpdf.Pdf, String)`

Sets the XMP creation date of a document.

`Cpdf.setModificationDateXMP(Cpdf.Pdf, String)`

Sets the XMP modification date of a document.

`Cpdf.getDateComponents(String, Int32@, Int32@, Int32@, Int32@, Int32@, Int32@, Int32@, Int32@)`

Returns the components from a PDF date string.

`Cpdf.dateStringOfComponents(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)`



Builds a PDF date string from individual components.

```
Cpdf.getPageRotation(Cpdf.Pdf, Int32)
```

Gets the viewing rotation for a given page.

```
Cpdf.hasBox(Cpdf.Pdf, Int32, String)
```

Returns true, if that page has the given box. E.g. "/CropBox".

```
Cpdf.getMediaBox(Cpdf.Pdf, Int32, Double@, Double@, Double@, Double@)
```

```
Cpdf.getCropBox(Cpdf.Pdf, Int32, Double@, Double@, Double@, Double@)
```

```
Cpdf.getTrimBox(Cpdf.Pdf, Int32, Double@, Double@, Double@, Double@)
```

```
Cpdf.getArtBox(Cpdf.Pdf, Int32, Double@, Double@, Double@, Double@)
```

```
Cpdf.getBleedBox(Cpdf.Pdf, Int32, Double@, Double@, Double@, Double@)
```

These functions get a box given the document, page number, min x, max x, min y, max y in points. Only succeeds if such a box exists, as checked by hasBox.

```
Cpdf.setMediabox(Cpdf.Pdf, List{Int32}, Double, Double, Double, Double)
```

```
Cpdf.setCropBox(Cpdf.Pdf, List{Int32}, Double, Double, Double, Double)
```

```
Cpdf.setTrimBox(Cpdf.Pdf, List{Int32}, Double, Double, Double, Double)
```

```
Cpdf.setArtBox(Cpdf.Pdf, List{Int32}, Double, Double, Double, Double)
```

```
Cpdf.setBleedBox(Cpdf.Pdf, List{Int32}, Double, Double, Double, Double)
```

These functions set a box given the document, page range, min x, max x, min y, max y in points.

```
Cpdf.markTrapped(Cpdf.Pdf)
```

Marks a document as trapped.

```
Cpdf.markUntrapped(Cpdf.Pdf)
```

Marks a document as untrapped.

```
Cpdf.markTrappedXMP(Cpdf.Pdf)
```

Marks a document as trapped in XMP metadata.

```
Cpdf.markUntrappedXMP(Cpdf.Pdf)
```

Marks a document as untrapped in XMP metadata.

`Cpdf.Layout`

Layouts

`Cpdf.Layout.SinglePage`

Single page

`Cpdf.Layout.OneColumn`

One column

`Cpdf.Layout.TwoColumnLeft`

Two column left

`Cpdf.Layout.TwoColumnRight`

Two column right

`Cpdf.Layout.TwoPageLeft`

Two page left

`Cpdf.Layout.TwoPageRight`

Two page right

`Cpdf.setPageLayout(Cpdf.Pdf, Cpdf.Layout)`

Sets the page layout for a document.

`Cpdf.PageMode`

Page modes

`Cpdf.PageMode.UseNone`

Use none

`Cpdf.PageMode.UseOutlines`

Use outlines

`Cpdf.PageMode.UseThumbs`

Use thumbs

`Cpdf.PageMode.UseOC`

Use OC

`Cpdf.PageMode.UseAttachments`

Use attachments

`Cpdf.setPageMode(Cpdf.Pdf, Cpdf.PageMode)`

Sets the page mode for a document.

`Cpdf.hideToolbar(Cpdf.Pdf, Boolean)`

Sets the hide toolbar flag.

`Cpdf.hideMenubar(Cpdf.Pdf, Boolean)`

Sets the hide menubar flag.

`Cpdf.hideWindowUi(Cpdf.Pdf, Boolean)`

Sets the hide window UI flag.

`Cpdf.fitWindow(Cpdf.Pdf, Boolean)`

Sets the fit window flag.

`Cpdf.centerWindow(Cpdf.Pdf, Boolean)`

Sets the center window flag.

`Cpdf.displayDocTitle(Cpdf.Pdf, Boolean)`

Sets the display doc title flag.

`Cpdf.openAtPage(Cpdf.Pdf, Boolean, Int32)`

Sets the PDF to open, possibly with zoom-to-fit, at the given page number.

`Cpdf.setMetadataFromFile(Cpdf.Pdf, String)`

Sets the XMP metadata of a document, given a file name.

`Cpdf.setMetadataFromByteArray(Cpdf.Pdf, Byte[])`

Sets the XMP metadata from an array of bytes.

`Cpdf.removeMetadata(Cpdf.Pdf)`

Removes the XMP metadata from a document.

`Cpdf.getMetadata(Cpdf.Pdf)`

Returns the XMP metadata from a document.

`Cpdf.createMetadata(Cpdf.Pdf)`

Builds fresh metadata as best it can from existing metadata in the document.

`Cpdf.setMetadataDate(Cpdf.Pdf, String)`

Sets the metadata date for a PDF. The date is given in PDF date format -- cpdf will convert it to XMP format. The date 'now' means now.

`Cpdf.PageLabelStyle`

Page label styles

`Cpdf.PageLabelStyle.DecimalArabic`

1, 2, 3...

`Cpdf.PageLabelStyle.UppercaseRoman`

I, II, III...

`Cpdf.PageLabelStyle.LowercaseRoman`

i, ii, iii...

`Cpdf.PageLabelStyle.UppercaseLetters`

A, B, C...

`Cpdf.PageLabelStyle.LowercaseLetters`

a, b, c...

`Cpdf.addPageLabels(Cpdf.Pdf, Cpdf.PageLabelStyle, String, Int32, List{Int32}, Boolean)`

Adds page labels. The prefix is prefix text for each label. The range is the page range the labels apply to. Offset can be used to shift the numbering up or down.

`Cpdf.removePageLabels(Cpdf.Pdf)`

Removes the page labels from the document.

`Cpdf.getPageLabelStringForPage(Cpdf.Pdf, Int32)`

Calculates the full label string for a given page, and returns it.

```
Cpdf.startGetPageLabels(Cpdf.Pdf)
```

```
Cpdf.getPageLabelStyle(Int32)
```

```
Cpdf.getPageLabelPrefix(Int32)
```

```
Cpdf.getPageLabelOffset(Int32)
```

```
Cpdf.getPageLabelRange(Int32)
```

```
Cpdf.endGetPageLabels
```



# Chapter 12

## File Attachments

```
cpdf -attach-file <filename> [-to-page <page number>] in.pdf -o out.pdf
cpdf -list-attached-files in.pdf
cpdf -remove-files in.pdf -o out.pdf
cpdf -dump-attachments in.pdf -o <directory>
```

PDF supports adding attachments (files of any kind, including other PDFs) to an existing file. The `cpdf` tool supports adding and removing *document-level attachments* — that is, ones which are associated with the document as a whole rather than with an individual page, and also *page-level attachments*, associated with a particular page.

### 12.1 Adding Attachments

To add an attachment, use the `-attach-file` operation. For instance,

```
cpdf -attach-file sheet.xls in.pdf -o out.pdf
```

attaches the Excel spreadsheet `sheet.xls` to the input file. If the file already has attachments, the new file is added to their number. You can specify multiple files to be attached by using `-attach-file` multiple times. They will be attached in the given order.

The `-to-page` option can be used to specify that the files will be attached to the given page, rather than at the document level. The `-to-page` option may be specified at most once.

### 12.2 Listing Attachments

To list all document- and page-level attachments, use the `-list-attached-files` operation. The page number and filename of each attachment is given, page 0 representing a document-level attachment.

```
$cpdf -list-attached-files 14psfonts.pdf
0 utility.ml
0 utility.mli
4 notes.xls
```

## 12.3 Removing Attachments

To remove all document-level and page-level attachments from a file, use the `-remove-files` operation:

```
cpdf -remove-files in.pdf -o out.pdf
```

## 12.4 Dumping Attachments to File

The `-dump-attachments` operation, when given a PDF file and a directory path as the output, will write each attachment under its filename (as displayed by `-list-attached-files` to that directory). The directory must exist prior to the call.

```
cpdf -dump-attachments in.pdf -o /home/fred/attachments
```

Unless the `-raw` option is given, the filenames are stripped of dubious special characters before writing. It is converted from unicode to 7 bit ASCII, and the following characters are removed, in addition to any character with ASCII code less than 32:

```
/ ? < > \ : * | " ^ + =
```



## .NET Interface

CHAPTER 12. File Attachments

`Cpdf.attachFile(String, Cpdf.Pdf)`

Attaches a file to the pdf. It is attached at document level.

`Cpdf.attachFileToPage(String, Cpdf.Pdf, Int32)`

Attaches a file, given its file name, pdf, and the page number to which it should be attached.

`Cpdf.attachFileFromMemory(Byte[], String, Cpdf.Pdf)`

Attaches data from memory, just like `attachFile`.

`Cpdf.attachFileToPageFromMemory(Byte[], String, Cpdf.Pdf, Int32)`

Attaches to a page from memory, just like `attachFileToPage`.

`Cpdf.removeAttachedFiles(Cpdf.Pdf)`

Removes all page- and document-level attachments from a document.

`Cpdf.startGetAttachments(Cpdf.Pdf)`

`Cpdf.numberGetAttachments`

Lists information about attachments. Call `startGetAttachments(pdf)` first, then `numberGetAttachments` to find out how many there are. Then `getAttachmentName` etc. to return each one  $0 \dots (n - 1)$ . Finally, call `endGetAttachments` to clean up.

`Cpdf.getAttachmentName(Int32)`

Gets the name of an attachment.

`Cpdf.getAttachmentPage(Int32)`

Gets the page number. 0 = document level.

`Cpdf.getAttachmentData(Int32)`

Gets the attachment data itself.

`Cpdf.endGetAttachments`

Cleans up after getting attachments.



# Chapter 13

## Working with Images

```
cpdf -extract-images in.pdf [<range>] [-im <path>] [-p2p <path>]
    [-dedup | -dedup-perpage] -o <path>
cpdf -image-resolution <minimum resolution> in.pdf [<range>]
```

### 13.1 Extracting images

Cpdf can extract the raster images to a given location. JPEG, JPEG2000 and JBIG2 images are extracted directly. Other images are written as PNGs, processed with either ImageMagick's "magick" command, or NetPBM's "pnmtopng" program, whichever is installed.

```
cpdf -extract-images in.pdf [<range>] [-im <path>] [-p2p <path>]
    [-dedup | -dedup-perpage] -o <path>
```

The `-im` or `-p2p` option is used to give the path to the external tool, one of which must be installed. The output specifier, e.g. `-o output/%%%` gives the number format for numbering the images. Output files are named serially from 0, and include the page number too. For example, output files might be called `output/000-p1.jpg`, `output/001-p1.png`, `output/002-p3.jpg` etc. Here is an example invocation:

```
cpdf -extract-images in.pdf -im magick -o output/%%%
```

The output directory must already exist. The `-dedup` option deduplicates images entirely; the `-dedup-perpage` option only per page.

### 13.2 Detecting Low-resolution Images

To list all images in the given range of pages which fall below a given resolution (in dots-per-inch), use the `-image-resolution` function:

```
cpdf -image-resolution 300 in.pdf [<range>]
```

```
2, /Im5, 531, 684, 149.935297, 150.138267  
2, /Im6, 184, 164, 149.999988, 150.458710  
2, /Im7, 171, 156, 149.999996, 150.579145  
2, /Im9, 65, 91, 149.999986, 151.071856  
2, /Im10, 94, 60, 149.999990, 152.284285  
2, /Im15, 184, 139, 149.960011, 150.672060  
4, /Im29, 53, 48, 149.970749, 151.616446
```

The format is *page number, image name, x pixels, y pixels, x resolution, y resolution*. The resolutions refer to the image's effective resolution at point of use (taking account of scaling, rotation etc).

### 13.3 Removing an Image

To remove a particular image, find its name using `-image-resolution` with a sufficiently high resolution (so as to list all images), and then apply the `-draft` and `-draft-remove-only` operations from Section 18.1.

## .NET Interface

CHAPTER 13. Images.

```
Cpdf.startGetImageResolution(Cpdf.Pdf, Double)
```

```
Cpdf.getImageResolutionPageNumber(Int32)
```

```
Cpdf.getImageResolutionImageName(Int32)
```

```
Cpdf.getImageResolutionXPixels(Int32)
```

```
Cpdf.getImageResolutionYPixels(Int32)
```

```
Cpdf.getImageResolutionXRes(Int32)
```

```
Cpdf.getImageResolutionYRes(Int32)
```

```
Cpdf.endGetImageResolution
```

Gets image data, including resolution at all points of use. Call `startGetImageResolution(pdf, min_required_resolution)` will begin the process of obtaining data on all image uses below `min_required_resolution`, returning the total number. So, to return all image uses, specify a very high `min_required_resolution`. Then, call the other functions giving a serial number `0..n - 1`, to retrieve the data. Finally, call `endGetImageResolution` to clean up.



# Chapter 14

## Fonts

```
cpdf -list-fonts in.pdf
cpdf -print-font-table <font name> -print-font-table-page <n> in.pdf
cpdf -copy-font fromfile.pdf -copy-font-page <int>
    -copy-font-name <name> in.pdf [<range>] -o out.pdf
cpdf -remove-fonts in.pdf -o out.pdf
cpdf -missing-fonts in.pdf
cpdf -embed-missing-fonts -gs <path to gs> in.pdf -o out.pdf
```

### 14.1 Listing Fonts

The `-list-fonts` operation prints the fonts in the document, one-per-line to standard output. For example:

```
1 /F245 /Type0 /Cleargothic-Bold /Identity-H
1 /F247 /Type0 /ClearGothicSerialLight /Identity-H
1 /F248 /Type1 /Times-Roman /WinAnsiEncoding
1 /F250 /Type0 /Cleargothic-RegularItalic /Identity-H
2 /F13 /Type0 /Cleargothic-Bold /Identity-H
2 /F16 /Type0 /Arial-ItalicMT /Identity-H
2 /F21 /Type0 /ArialMT /Identity-H
2 /F58 /Type1 /Times-Roman /WinAnsiEncoding
2 /F59 /Type0 /ClearGothicSerialLight /Identity-H
2 /F61 /Type0 /Cleargothic-BoldItalic /Identity-H
2 /F68 /Type0 /Cleargothic-RegularItalic /Identity-H
3 /F47 /Type0 /Cleargothic-Bold /Identity-H
3 /F49 /Type0 /ClearGothicSerialLight /Identity-H
3 /F50 /Type1 /Times-Roman /WinAnsiEncoding
3 /F52 /Type0 /Cleargothic-BoldItalic /Identity-H
```

```

3 /F54 /Type0 /TimesNewRomanPS-BoldItalicMT /Identity-H
3 /F57 /Type0 /Cleargothic-RegularItalic /Identity-H
4 /F449 /Type0 /Cleargothic-Bold /Identity-H
4 /F451 /Type0 /ClearGothicSerialLight /Identity-H
4 /F452 /Type1 /Times-Roman /WinAnsiEncoding

```

The first column gives the page number, the second the internal unique font name, the third the type of font (Type1, TrueType etc), the fourth the PDF font name, the fifth the PDF font encoding.

## 14.2 Listing characters in a font

We can use `cpdf` to find out which characters are available in a given font, and to print the map between character codes, unicode codepoints, and Adobe glyph names. This is presently a best-effort service, and does not cover all font/encoding types.

We find the name of the font by using `-list-fonts`:

```

$ ./cpdf -list-fonts cpdfmanual.pdf 1
1 /F46 /Type1 /XYPLPB+NimbusSanL-Bold
1 /F49 /Type1 /MCBERL+URWPalladioL-Roma

```

We may then print the table, giving either the font's name (e.g `/F46`) or basename (e.g `/XYPLPB+NimbusSanL-Bold`):

```

$ ./cpdf -print-font-table /XYPLPB+NimbusSanL-Bold
-print-font-table-page 1 cpdfmanual.pdf
67 = U+0043 (C - LATIN CAPITAL LETTER C) = /C
68 = U+0044 (D - LATIN CAPITAL LETTER D) = /D
70 = U+0046 (F - LATIN CAPITAL LETTER F) = /F
71 = U+0047 (G - LATIN CAPITAL LETTER G) = /G
76 = U+004C (L - LATIN CAPITAL LETTER L) = /L
80 = U+0050 (P - LATIN CAPITAL LETTER P) = /P
84 = U+0054 (T - LATIN CAPITAL LETTER T) = /T
97 = U+0061 (a - LATIN SMALL LETTER A) = /a
99 = U+0063 (c - LATIN SMALL LETTER C) = /c
100 = U+0064 (d - LATIN SMALL LETTER D) = /d
101 = U+0065 (e - LATIN SMALL LETTER E) = /e
104 = U+0068 (h - LATIN SMALL LETTER H) = /h
105 = U+0069 (i - LATIN SMALL LETTER I) = /i
108 = U+006C (l - LATIN SMALL LETTER L) = /l
109 = U+006D (m - LATIN SMALL LETTER M) = /m
110 = U+006E (n - LATIN SMALL LETTER N) = /n
111 = U+006F (o - LATIN SMALL LETTER O) = /o
112 = U+0070 (p - LATIN SMALL LETTER P) = /p
114 = U+0072 (r - LATIN SMALL LETTER R) = /r
115 = U+0073 (s - LATIN SMALL LETTER S) = /s
116 = U+0074 (t - LATIN SMALL LETTER T) = /t

```



The first column is the character code, the second the Unicode codepoint, the character itself and its Unicode name, and the third the Adobe glyph name.

### 14.3 Copying Fonts

In order to use a font other than the standard 14 with `-add-text`, it must be added to the file. The font source PDF is given, together with the font's resource name on a given page, and that font is copied to all the pages in the input file's range, and then written to the output file.

The font is named in the output file with its basefont name, so it can be easily used with `-add-text`.

For example, if the file `fromfile.pdf` has a font `/GHLIGA+c128` with the name `/F10` on page 1 (this information can be found with `-list-fonts`), the following would copy the font to the file `in.pdf` on all pages, writing the output to `out.pdf`:

```
cpdf -copy-font fromfile.pdf -copy-font-name /F10
      -copy-font-page 1 in.pdf -o out.pdf
```

Text in this font can then be added by giving `-font /GHLIGA+c128`. Be aware that due to the vagaries of PDF font handling concerning which characters are present in the source font, not all characters may be available, or `cpdf` may not be able to work out the conversion from UTF8 to the font's own encoding. You may add `-raw` to the command line to avoid any conversion, but the encoding (mapping from input codes to glyphs) may be non-obvious and require knowledge of the PDF format to divine.

### 14.4 Removing Fonts

To remove embedded fonts from a document, use `-remove-fonts`. PDF readers will substitute local fonts for the missing fonts. The use of this function is only recommended when file size is the sole consideration.

```
cpdf -remove-fonts in.pdf -o out.pdf
```

### 14.5 Missing Fonts

The `-missing-fonts` operation lists any unembedded fonts in the document, one per line.

```
cpdf -missing-fonts in.pdf
```

The format is

```
Page number, Name, Subtype, Basefont, Encoding
```

The operation `-embed-missing-fonts` will process the file with `gs` (which must be installed) to embed missing fonts (where found):

```
cpdf -embed-missing-fonts -gs gs in.pdf -o out.pdf
```

## .NET Interface

CHAPTER 14. Fonts.

`Cpdf.startGetFontInfo(Cpdf.Pdf)`

`Cpdf.numberFonts`

`Cpdf.getFontPage(Int32)`

`Cpdf.getFontName(Int32)`

`Cpdf.getFontType(Int32)`

`Cpdf.getFontEncoding(Int32)`

`Cpdf.endGetFontInfo`

Retrieves font information. First, call `startGetFontInfo(pdf)`. Now call `numberFonts` to return the number of fonts. For each font, call one or more of `getPage`, `getName`, `getType`, and `getEncoding` giving a serial number `0..n - 1` to return information. Finally, call `endGetFontInfo` to clean up.

`Cpdf.removeFonts(Cpdf.Pdf)`

Removes all font data from a file.

`Cpdf.copyFont(Cpdf.Pdf, Cpdf.Pdf, List{Int32}, Int32, String)`

Copies the given font from the given page in the 'from' PDF to every page in the 'to' PDF. The new font is stored under its font name.



# Chapter 15

## PDF and JSON

```
cpdf in.pdf -output-json -o out.json
  [-output-json-parse-content-streams]
  [-output-json-no-stream-data]
  [-output-json-decompress-streams]
  [-output-json-clean-strings]
cpdf -j in.json -o out.pdf
```

In addition to reading and writing PDF files in the original Adobe format, `cpdf` can read and write them in its own CPDFJSON format, for somewhat easier extraction of information, modification of PDF files, and so on.

### 15.1 Converting PDF to JSON

We convert a PDF file to JSON format like this:

```
cpdf -output-json in.pdf -o out.json
```

The resultant JSON file is an array of arrays containing an object number followed by an object, one for each object in the file and two special ones:

- Object -1: CPDF's own data with the PDF version number, CPDF JSON format number, and flags used when writing (which may be required when reading):
  - /CPDFJSONformatversion (CPDFJSON integer (see below), currently 2)
  - /CPDFJSONcontentparsed (boolean, true if content streams have been parsed)
  - /CPDFJSONstreamdataincluded (boolean, true if stream data included. Cannot round-trip if false).
  - /CPDFJSONmajorpdfversion (CPDFJSON integer)

- /CPDFJSONminorpdfversion (CPDFJSON integer)

- Object 0: The PDF's trailer dictionary
- Objects 1..n: The PDF's objects.

Objects are formatted thus:

- PDF arrays, dictionaries, booleans, and strings are the same as in JSON.
- Integers are written as {"I": 0}
- Floats are written as {"F": 0.0}
- Names are written as {"N": "/Pages"}
- Indirect references are integers
- Streams are {"S": [dict, data]}
- Strings are converted to JSON string format in a way which, when reversed, results in the original string.

Here is an example of the output for a small PDF:

```
[
  [
    -1,
    { "/CPDFJSONformatversion": { "I": 2 },
      "/CPDFJSONcontentparsed": false,
      "/CPDFJSONstreamdataincluded": true,
      "/CPDFJSONmajorpdfversion": { "I": 1 },
      "/CPDFJSONminorpdfversion": { "I": 1 } }
  ],
  [
    0,
    { "/Size": { "I": 4 }, "/Root": 4,
      "/ID" : [ <elided>, <elided> ] } ],
  [
    1, { "/Type": { "N": "/Pages" }, "/Kids": [ 3 ], "/Count": { "I": 1 } }
  ],
  [
    2,
    { "S": [{ "/Length": { "I": 49 } },
      "1 0 0 1 50 770 cm BT/F0 36 Tf(Hello, World!)Tj ET" ] }
  ],
  [
    3, { "/Type": { "N": "/Page" }, "/Parent": 1,
      "/Resources": {
        "/Font": {
          "/F0": {
            "/Type": { "N": "/Font" },
```

```

        "/Subtype": { "N": "/Type1" },
        "/BaseFont": { "N": "/Times-Italic" }
    }
},
"/MediaBox":
  [{ "I": 0 }, { "I": 0 },
   { "F": 595.2755905510001 }, { "F": 841.88976378 }],
"/Rotate": { "I": 0 },
"/Contents": [ 2 ] } ],
[
  4, { "/Type": { "N": "/Catalog" }, "/Pages": 1 } ]
]

```

The option `-output-json-parse-content-streams` will also convert content streams to JSON, so our example content stream will be expanded:

```

2, {
  "S": [
    {}, [
      [
        { "F": 1.0 }, { "F": 0.0 }, { "F": 0.0 }, { "F": 1.0 }, { "F": 50.0 }, {
          "F": 770.0 }, "cm" ], [ "BT" ], [ "/F0", { "F": 36.0 }, "Tf" ], [
            "Hello, World!", "Tj" ], [ "ET" ] ]
      ] } ], [

```

The option `-output-json-no-stream-data` simply elides the stream data instead, leading to much smaller JSON files.

The option `-output-json-decompress-streams` keeps the streams intact, and decompresses them.

The option `-output-json-clean-strings` converts any UTF16BE strings with no high bytes to PDFDocEncoding prior to output, so that editing them is easier.

## 15.2 Converting JSON to PDF

We can load a JSON PDF file with the `-j` option in place of a PDF file anywhere in a normal `cpdf` command. A range may be applied, just like any other file.

```
cpdf -j in.json -o out.pdf
```

It is not required that `/Length` entries in CPDFJSON stream dictionaries be correctly updated when the JSON file is edited: `cpdf` will fix them when loading.

## .NET Interface

CHAPTER 15. PDF and JSON

```
Cpdf.outputJSON(String, Boolean, Boolean, Boolean, Cpdf.Pdf)
```

Outputs a PDF in JSON format to the given filename. If `parse_content` is true, page content is parsed. If `no_stream_data` is true, all stream data is suppressed entirely.

```
Cpdf.outputJSONMemory(Cpdf.Pdf, Boolean, Boolean, Boolean)
```

Like `outputJSON`, but it writes to a byte array in memory.

```
Cpdf.fromJSON(String)
```

Loads a PDF from a JSON file given its filename.

```
Cpdf.fromJSONMemory(Byte[])
```

Loads a PDF from a JSON file in memory



## Chapter 16

# Optional Content Groups

```
cpdf -ocg-list in.pdf
cpdf -ocg-rename -ocg-rename-from <a> -ocg-rename-to <b> in.pdf -o out.pdf
cpdf -ocg-order-all in.pdf -o out.pdf
cpdf -ocg-coalesce-on-name in.pdf -o out.pdf
```

In a PDF file, optional content groups are used to group graphical elements together, so they may appear or not, depending on the preference of the user. They are similar in some ways to layers in graphics illustration programs.

```
cpdf -ocg-list in.pdf
```

List the optional content groups in the PDF, one per line, to standard output. UTF8.

```
cpdf -ocg-rename -ocg-rename-from <a> -ocg-rename-to <b> in.pdf -o out.pdf
```

Rename an optional content group.

```
cpdf -ocg-coalesce-on-name in.pdf -o out.pdf
```

Coalesce optional content groups. For example, if we merge or stamp two files both with an OCG called "Layer 1", we will have two different optional content groups. Running `-ocg-coalesce-on-name` will merge the two into a single optional content group.

```
cpdf -ocg-order-all in.pdf -o out.pdf
```

Ensure that every optional content group appears in the order list.

## .NET Interface

CHAPTER 16. Optional Content Groups

`Cpdf.startGetOCGList(Cpdf.Pdf)`

Begins retrieving optional content group names. The serial number  $0..n - 1$  is returned.

`Cpdf.OCGListEntry(Int32)`

Retrieves an OCG name, given its serial number  $0..n - 1$ .

`Cpdf.endGetOCGList`

Ends retrieval of optional content group names.

`Cpdf.OCGRename(Cpdf.Pdf, String, String)`

Renames an optional content group.

`Cpdf.OCGOrderAll(Cpdf.Pdf)`

Ensures that every optional content group appears in the OCG order list.

`Cpdf.OCGCoalesce(Cpdf.Pdf)`

Coalesces optional content groups. For example, if we merge or stamp two files both with an OCG called "Layer 1", we will have two different optional content groups. This function will merge the two into a single optional content group.

# Chapter 17

## Creating New PDFs

```
cpdf -create-pdf [-create-pdf-pages <n>]
             [-create-pdf-papersize <paper size>] -o out.pdf
cpdf -typeset <text file> [-create-pdf-papersize <size>]
             [-font <font>] [-font-size <size>] -o out.pdf
```

### 17.1 A new blank PDF

We can build a new PDF file, given a number of pages and a paper size. The default is one page, A4 portrait.

```
cpdf -create-pdf -create-pdf-pages 20
      -create-pdf-papersize usletterportrait -o out.pdf
```

The standard paper sizes are listed in Section 3.1, or you may specify the width and height directly, as described in the same chapter.

### 17.2 Convert a text file to PDF

A basic text to PDF convertor is included in `cpdf`. It takes a UTF8 text file (ASCII is a subset of UTF8) and typesets it ragged-right, splitting on whitespace. Both Windows and Unix line endings are allowed.

```
cpdf -typeset file.txt -create-pdf-papersize a3portrait
      -font Courier -font-size 10 -o out.pdf
```

The standard paper sizes are listed in Section 3.1, or you may specify the width and height directly, as described in the same chapter. The standard fonts are listed in chapter 8. The default font is TimesRoman and the default size is 12.

## .NET Interface

CHAPTER 17. Creating New PDFs

`Cpdf.blankDocument(Double, Double, Int32)`

Creates a blank document with pages of the given width (in points), height (in points), and number of pages.

`Cpdf.blankDocumentPaper(Cpdf.Papersize, Int32)`

Makes a blank document given a page size and number of pages.

`Cpdf.textToPDF(Double, Double, Cpdf.Font, Double, String)`

Typesets a UTF8 text file ragged right on a page of size  $w * h$  in points in the given font and font size.

`Cpdf.textToPDFPaper(Cpdf.Papersize, Cpdf.Font, Double, String)`

Typesets a UTF8 text file ragged right on a page of the given size in the given font and font size.

# Chapter 18

## Miscellaneous

```
cpdf -draft [-boxes] [-draft-remove-only <n>] in.pdf [<range>] -o out.pdf
cpdf -remove-all-text in.pdf [<range>] -o out.pdf
cpdf -blacktext in.pdf [<range>] -o out.pdf
cpdf -blacklines in.pdf [<range>] -o out.pdf
cpdf -blackfills in.pdf [<range>] -o out.pdf
cpdf -thinline <minimum thickness> in.pdf [<range>] -o out.pdf
cpdf -clean in.pdf -o out.pdf
cpdf -set-version <version number> in.pdf -o out.pdf
cpdf -copy-id-from source.pdf in.pdf -o out.pdf
cpdf -remove-id in.pdf -o out.pdf
cpdf -list-spot-colors in.pdf
cpdf -print-dict-entry <key> in.pdf
cpdf -remove-dict-entry <key> [-dict-entry-search <term>]
    in.pdf -o out.pdf
cpdf -replace-dict-entry <key> -replace-dict-entry-value <value>
    [-dict-entry-search <term>] in.pdf -o out.pdf
cpdf -remove-clipping [<range>] in.pdf -o out.pdf
```

### 18.1 Draft Documents

The `-draft` operation removes bitmap (photographic) images from a file, so that it can be printed with less ink. Optionally, the `-boxes` option can be added, filling the spaces left blank with a crossed box denoting where the image was. This is not guaranteed to be fully visible in all cases (the bitmap may have been partially covered by vector objects or clipped in the original). For example:

```
cpdf -draft -boxes in.pdf -o out.pdf
```

To remove a single image only, specify `-draft-remove-only`, giving the name of the image obtained by a call to `-image-resolution` as described in Section 13.2 and giving the appropriate page. For example:

```
cpdf -draft -boxes -draft-remove-only "/Im1" in.pdf 7 -o out.pdf
```

To remove text instead of images, use the `-remove-all-text` operation:

```
cpdf -remove-all-text in.pdf -o out.pdf
```

## 18.2 Blackening Text, Lines and Fills

Sometimes PDF output from an application (for instance, a web browser) has text in colors which would not print well on a grayscale printer. The `-blacktext` operation blackens all text on the given pages so it will be readable when printed.

This will not work on text which has been converted to outlines, nor on text which is part of a form.

```
cpdf -blacktext in.pdf -o out.pdf
```

The `-blacklines` operation blackens all lines on the given pages.

```
cpdf -blacklines in.pdf -o out.pdf
```

The `-blackfills` operation blackens all fills on the given pages.

```
cpdf -blackfills in.pdf -o out.pdf
```

Contrary to their names, all these operations can use another color, if specified with `-color`.

## 18.3 Hairline Removal

Quite often, applications will use very thin lines, or even the value of 0, which in PDF means "The thinnest possible line on the output device". This might be fine for on-screen work, but when printed on a high resolution device, such as by a commercial printer, they may be too faint, or disappear altogether. The `-thinline` operation prevents this by changing all lines thinner than `<minimal thickness>` to the given thickness. For example:

```
cpdf -thinlines 0.2mm in.pdf [<range>] -o out.pdf
```

Thicken all lines less than 0.2mm to that value.

## 18.4 Garbage Collection

Sometimes incremental updates to a file by an application, or bad applications can leave data in a PDF file which is no longer used. This function removes that unneeded data.

```
cpdf -clean in.pdf -o out.pdf
```

## 18.5 Change PDF Version Number

To change the pdf version number, use the `-set-version` operation, giving the part of the version number after the decimal point. For example:

```
cpdf -set-version 4 in.pdf -o out.pdf
```

Change file to PDF 1.4.

This does not alter any of the actual data in the file — just the supposed version number. For PDF versions starting with 2 add ten to the number. For example, for PDF version 2.0, use `-set-version 10`.

## 18.6 Copy ID

The `-copy-id-from` operation copies the ID from the given file to the input, writing to the output.

```
cpdf -copy-id-from source.pdf in.pdf -o out.pdf
```

Copy the id from `source.pdf` to the contents of `in.pdf`, writing to `out.pdf`.

If there is no ID in the source file, the existing ID is retained. You cannot use `-recrypt` with `-copy-id-from`.

## 18.7 Remove ID

The `-remove-id` operation removes the ID from a document.

```
cpdf -remove-id in.pdf -o out.pdf
Remove the ID from in.pdf, writing to out.pdf.
```

You cannot use `-recrypt` with `-remove-id`.

## 18.8 List Spot Colours

This operation lists the name of any “separation” color space in the given PDF file.

```
cpdf -list-spot-colors in.pdf
List the spot colors, one per line in in.pdf, writing to stdout.
```

## 18.9 PDF Dictionary Entries

This is for editing data within the PDF’s internal representation. Use with caution.

To print a dictionary entry:

```
cpdf -print-dict-entry /URI in.pdf -o out.pdf
Print all URLs in annotation hyperlinks in.pdf.
```

To remove a dictionary entry:

```
cpdf -remove-dict-entry /One in.pdf -o out.pdf
Remove the entry for /One in every dictionary in.pdf, writing to out.pdf.

cpdf -remove-dict-entry /One -dict-entry-search "1" in.pdf -o out.pdf
Replace the entry for /One in every dictionary in.pdf if the key’s value is the given value, writing to out.pdf.
```

To replace a dictionary entry:

```
cpdf -replace-dict-entry /One -replace-dict-entry-value "2"
in.pdf -o out.pdf
Remove the entry for /One in every dictionary in.pdf, writing to out.pdf.

cpdf -replace-dict-entry /One -dict-entry-search "1"
-replace-dict-entry-value "2" in.pdf -o out.pdf
```



Remove the entry for /One in every dictionary in .pdf if the key's value is the given value, writing to out.pdf.

## 18.10 Removing Clipping

The `-remove-clipping` operation removes any clipping paths on given pages from the file.

```
cpdf -remove-clipping in.pdf -o out.pdf
```

Remove clipping paths in in.pdf, writing to out.pdf.

## .NET Interface

CHAPTER 18. Miscellaneous

`Cpdf.draft(Cpdf.Pdf, List{Int32}, Boolean)`

Removes images on the given pages, replacing them with crossed boxes if 'boxes' is true.

`Cpdf.removeAllText(Cpdf.Pdf, List{Int32})`

Removes all text from the given pages in a given document.

`Cpdf.blackText(Cpdf.Pdf, List{Int32})`

Blackens all text on the given pages.

`Cpdf.blackLines(Cpdf.Pdf, List{Int32})`

Blackens all lines on the given pages.

`Cpdf.blackFills(Cpdf.Pdf, List{Int32})`

Blackens all fills on the given pages.

`Cpdf.thinLines(Cpdf.Pdf, List{Int32}, Double)`

Thickens every line less than `min_thickness` to `min_thickness`. Thickness given in points.

`Cpdf.copyId(Cpdf.Pdf, Cpdf.Pdf)`

Copies the /ID from one document to another.

`Cpdf.removeId(Cpdf.Pdf)`

Removes a document's /ID.

`Cpdf.setVersion(Cpdf.Pdf, Int32)`

Sets the minor version number of a document.

`Cpdf.setFullVersion(Cpdf.Pdf, Int32, Int32)`

Sets the full version number of a document.

`Cpdf.removeDictEntry(Cpdf.Pdf, String)`

Removes any dictionary entry with the given key anywhere in the document.

```
Cpdf.removeDictEntrySearch(Cpdf.Pdf, String, String)
```

Removes any dictionary entry with the given key whose value matches the given search term.

```
Cpdf.replaceDictEntry(Cpdf.Pdf, String, String)
```

Replaces the value associated with the given key.

```
Cpdf.replaceDictEntrySearch(Cpdf.Pdf, String, String, String)
```

Replaces the value associated with the given key if the existing value matches the search term.

```
Cpdf.removeClipping(Cpdf.Pdf, List{Int32})
```

Removes all clipping from pages in the given range.

```
Cpdf.getDictEntries(Cpdf.Pdf, String)
```

Returns a JSON array containing any and all values associated with the given key, and fills in its length.



# Appendix A

## Dates

### A.1 PDF Date Format

Dates in PDF are specified according to the following format:

D : YYYYYMMDDHHmmSSOHH ' mm '

where:

- YYYY is the year;
- MM is the month;
- DD is the day (01-31);
- HH is the hour (00-23);
- mm is the minute (00-59);
- SS is the second (00-59);
- O is the relationship of local time to Universal Time (UT), denoted by '+', '-' or 'Z';
- HH is the absolute value of the offset from UT in hours (00-23);
- mm is the absolute value of the offset from UT in minutes (00-59).

A contiguous prefix of the parts above can be used instead, for lower accuracy dates. For example:

D : 2014 (2014)

D : 20140103 (3rd January 2014)

`D:201401031854-08'00' (3rd January 2014, 6:54PM, US Pacific Standard Time)`

## A.2 XMP Metadata Date Format

These are the possible data formats for `-set-metadata-date`:

YYYY

YYYY-MM

YYYY-MM-DD

YYYY-MM-DDThh:mmTZD

YYYY-MM-DDThh:mm:ssTZD

where:

YYYY	year
MM	month (01 = Jan)
DD	day of month (01 to 31)
hh	hour (00 to 23)
mm	minute (00 to 59)
ss	second (00 to 59)
TZD	time zone designator (Z or +hh:mm or -hh:mm)

# Index

- AND, 5
- annotations
  - copying, 72
  - listing, 71
  - removing, 72
- attachments, 91
  - adding, 91
  - dumping to file, 92
  - listing, 91
  - removing, 92
- bates numbers, 57
- blacken
  - fills, 114
  - lines, 114
  - text, 114
- blank pages
  - inserting, 67
- bookmarks, 43
  - adding, 45
  - listing, 43
  - opening at level, 46
  - removing, 45
- collation, 20
- color, 59
- compressing, 39
- control file, 9
- Create, 111
- create new PDF, 111
- creator, 6
- crop pages, 26
- date, 56
  - defined, 121
- decompressing, 39
- decryption, 3, 35
- dictionary
  - print entry, 116
  - remove entry, 116
  - replace entry, 116
- document information, 75
- document outline, 43
- draft, 113
- encryption, 21, 35
- error handling, 9
- file ID, 6
  - copy, 115
  - remove, 115
- flip pages, 26
- font, 58
  - embedding, 10
  - listing, 99
  - print table for, 100
- garbage collection, 115
- hairline removal, 114
- imposition, 68
- input files, 1
- input range, 2
- JSON, 105
  - add bookmarks from, 43
  - input from, 105
  - list annotations as, 71
  - list bookmarks as, 43
  - output to, 105
- linearization, 6
- malformed file, 8
- media box, 26
- merging, 19

- metadata, 75
  - XMP, 78
- object stream, 7
- optional content group, 109
- outline text, 60
- output files, 1
- owner password, 3
- page
  - duplicate, 3
  - labels, 80
  - layout, 78
  - mode, 79
  - numbers, 57
  - range, 2
  - size, 24
- pages
  - collate, 20
- password, 3
- presentations, 51
- printer's marks, 28
- producer, 6
- range, 2
- removing text, 56
- reversing, 2
- rotate
  - contents, 26
  - pages, 25
- scale pages, 24
- shift page contents, 25
- splitting, 20
  - on bookmarks, 20
- spot colour, 116
- squeeze, 40
- stamp text, 56
- standard input, 4
- standard output, 4
- text
  - convert to PDF, 111
  - encodings, 10
- time, 56
- trim marks, 28
- two-up, 68
- units, 5
- user password, 3
- version number, 6, 115
- watermarks, 55
- XMP metadata, 78