Coherent PDF Library (libcpdf)

Developer's Manual

Version 2.2 (January 2017)

Coherent Graphics Ltd

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

Contents

Note

The chapters are numbered to be the same as those in the manual for the Coherent PDF Command Line Tools (cpdfmanual.pdf). However, some of the content has moved where circumstances dictate – for example, encryption is wholly described in Chapter 1 rather than Chapter 4.

All functions in cpdflib take or return UTF8 text - the command line tool's other modes, ″Raw″ and ″Stripped″ are not supported.

Installation

The Coherent PDF Library is provided either in compiled form (the archive file `libcpdf.a` and the library header `cpdflibwrapper.h`), or as source code. Instructions for building from source are included in the distribution.

Linux, Unix and OS/X Place `libcpdf.a, libbigarray.a and libunix.a` somewhere suitable. Instruct your C linker to link with them. Place `cpdflibwrapper.h` somewhere suitable and instuct your C compiler to search for headers there. The library is now ready for use.

Microsoft Windows Place `libcpdf.dll` somewhere suitable and instruct your C compiler to link with it. Place `cpdflibwrapper.h` somewhere suitable and instuct your C compiler to search for headers there. The library is now ready for use.

0 Preliminaries

```
/* A C wrapper to cpdf PDF tools library. Free for non-commercial use. See LICENSE
for details. To purchase a license, please visit http://www.coherentpdf.com/

Text arguments and results are in UTF8. */
```

```
/* CHAPTER 0. Preliminaries */

/* The function cpdf_startup(argv) must be called before using the library. */
void cpdf_startup (char **);

/* Return the version of the cpdflib library as a string */
char *cpdf_version ();

/* 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. These functions set the mode globally. */
void cpdf_setFast ();
void cpdf_setSlow ();

/* Undocumented. */
void cpdf_setDemo (int);

/* Errors. cpdf_lastError and cpdf_lastErrorString hold information about the
 * last error to have occurred. They should be consulted after each call. If
 * cpdf_lastError is non-zero, there was an error, and cpdf_lastErrorString
 * gives details. If cpdf_lastError is zero, there was no error on the most
 * recent cpdf call. */
extern int cpdf_lastError;
extern char *cpdf_lastErrorString;

/* cpdf_clearError clears the current error state. */
void cpdf_clearError (void);

/* cpdf_onExit is a debug function which prints some information about
 * resource usage. This can be used to detect if PDFs or ranges are being
 * deallocated properly. */
void cpdf_onExit (void);
```

1 Basics

```
/* CHAPTER 1. Basics */

/* cpdf_fromFile(filename, userpw) loads a PDF file from a given file. Supply a
 * user password (possibly blank) in case the file is encypted. It won't be
 * decrypted, but sometimes the password is needed just to load the file. */
int cpdf_fromFile (const char[], const char[]);

/* cpdf_fromFileLazy(pdf, userpw) 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 encypted. It won't be decrypted, but
 * sometimes the password is needed just to load the file. */
int cpdf_fromFileLazy (const char[], const char[]);

/* cpdf_fromMemory(data, length, userpw) loads a file from memory, given a
 * pointer and a length, and the user password. */
int cpdf_fromMemory (void *, int, const char[]);

/* cpdf_fromMemory(data, length, userpw) loads a file from memory, given a
 * pointer and a length, and the user password, but lazily like
 * cpdf_fromFileLazy. */
int cpdf_fromMemoryLazy (void *, int, const char[]);

/* cpdf_blankDocument(width, height, num_pages) creates a blank document with
 * pages of the given width (in points), height (in points), and number of
 * pages. */
int cpdf_blankDocument (double, double, int);

/* Standard page sizes. */
enum cpdf_papersize
{
  cpdf_a0portrait, /* A0 portrait */
  cpdf_a1portrait, /* A1 portrait */
  cpdf_a2portrait, /* A2 portrait */
  cpdf_a3portrait, /* A3 portrait */
  cpdf_a4portrait, /* A4 portrait */
  cpdf_a5portrait, /* A5 portrait */
  cpdf_a0landscape, /* A0 landscape */
  cpdf_a1landscape, /* A1 landscape */
  cpdf_a2landscape, /* A2 landscape */
  cpdf_a3landscape, /* A3 landscape */
  cpdf_a4landscape, /* A4 landscape */
```

```
  cpdf_a5landscape, /* A5 landscape */
  cpdf_usletterportrait, /* US Letter portrait */
  cpdf_usletterlandscape, /* US Letter landscape */
  cpdf_uslegalportrait, /* US Legal portrait */
  cpdf_uslegallandscape /* US Legal landscape */
};

/* cpdf_blankDocumentPaper(papersize, num_pages) makes a blank document given a
 * page size and number of pages. */
int cpdf_blankDocumentPaper (enum cpdf_papersize, int);

/* Remove a PDF from memory, given its number. */
void cpdf_deletePdf (int);

/* Calling cpdf_replacePdf(a, b) places PDF b under number a. Original a and b are
 * no longer available. */
void cpdf_replacePdf (int, int);

/* To enumerate the list of currently allocated PDFs, call
 * cpdf_startEnumeratePDFs which gives the number, n, of PDFs allocated, then
 * cpdf_enumeratePDFsInfo and cpdf_enumeratePDFsKey with index numbers from
 * 0...(n - 1). Call cpdf_endEnumeratePDFs to clean up. */
int cpdf_startEnumeratePDFs (void);
int cpdf_enumeratePDFsKey (int);
char *cpdf_enumeratePDFsInfo (int);
void cpdf_endEnumeratePDFs (void);

/* Convert a figure in centimetres to points (72 points to 1 inch) */
double cpdf_ptOfCm (double);

/* Convert a figure in millimetres to points (72 points to 1 inch) */
double cpdf_ptOfMm (double);

/* Convert a figure in inches to points (72 points to 1 inch) */
double cpdf_ptOfIn (double);

/* Convert a figure in points to centimetres (72 points to 1 inch) */
double cpdf_cmOfPt (double);

/* Convert a figure in points to millimetres (72 points to 1 inch) */
double cpdf_mmOfPt (double);

/* Convert a figure in points to inches (72 points to 1 inch) */
double cpdf_inOfPt (double);

/* A page range is a list of page numbers used to restrict operations to
 * certain pages. A page specification is a textual description of a page
 * range, such as "1-12,18-end". Here is the syntax:
 *
 * o A dash (-) defines ranges, e.g. 1-5 or 6-3.
 *
 * o A comma (,) allows one to specify several ranges, e.g. 1-2,4-5.
 *
 * o The word end represents the last page number.
```

```
 *
 * o 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.
 *
 * o 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 cpdf_upright first, if
 * required). A page with equal width and height is considered neither
 * portrait nor landscape.
 *
 * o The word reverse is the same as end-1.
 *
 * o The word all is the same as 1-end.
 *
 * o A range must contain no spaces.
 *
 * o 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. */

/* cpdf_parsePagespec(pdf, range) 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). */
int cpdf_parsePagespec (int, const char[]);

/* cpdf_validatePagespec(range) validates a page specification so far as is
 * possible in the absence of the actual document */
int cpdf_validatePagespec (const char[]);

/* cpdf_stringOfPagespec(pdf, range) 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" */
char *cpdf_stringOfPagespec (int, int);

/* cpdf_blankRange() creates a range with no pages in. */
int cpdf_blankRange (void);

/* cpdf_deleteRange(range) deletes a range. */
void cpdf_deleteRange (int);

/* cpdf_range(from, to) build a range from one page to another inclusive. For example,
 * cpdf_range(3,7) gives the range 3,4,5,6,7 */
int cpdf_range (int, int);

/* cpdf_all(pdf) is the range containing all the pages in a given document. */
int cpdf_all (int);

/* cpdf_even(range) makes a range which contains just the even pages of another
 * range */
int cpdf_even (int);

/* cpdf_odd(range) makes a range which contains just the odd pages of another
 * range */
```

```
int cpdf_odd (int);

/* cpdf_rangeUnion(a, b) makes the union of two ranges giving a range containing t
 * pages in range a and range b. */
int cpdf_rangeUnion (int, int);

/* cpdf_difference(a, b) makes the difference of two ranges, giving a range
 * containing all the pages in a except for those which are also in b. */
int cpdf_difference (int, int);

/* cpdf_removeDuplicates(range) deduplicates a range, making a new one. */
int cpdf_removeDuplicates (int);

/* cpdf_rangeLenght gives the number of pages in a range. */
int cpdf_rangeLength (int);

/* cpdf_rangeGet(range, n) gets the page number at position n in a range, where
 * n runs from 0 to rangeLength - 1. */
int cpdf_rangeGet (int, int);

/* cpdf_rangeAdd(range, page) adds the page to a range, if it is not already
 * there. */
int cpdf_rangeAdd (int, int);

/* cpdf_isInRange(range, page) returns true if the page is in the range, false
 * otherwise. */
int cpdf_isInRange (int, int);

/* cpdf_pages(pdf) returns the number of pages in a PDF. */
int cpdf_pages (int);

/* cpdf_pagesFast(password, filename) returns the number of pages in a given
 * PDF, with given user encryption password. It tries to do this as fast as
 * possible, without loading the whole file. */
int cpdf_pagesFast (const char[], const char[]);

/* cpdf_toFile (pdf, filename, linearize, make_id) writes the file to a given
 * filename. If linearize is true, it will be linearized. If make_id is true,
 * it will be given a new ID. */
void cpdf_toFile (int, const char[], int, int);

/* cpdf_toFile (pdf, filename, linearize, make_id, preserve_objstm,
 * generate_objstm, compress_objstm) 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 discarded. */
void cpdf_toFileExt (int, const char[], int, int, int, int, int);

/* Given a buffer of the correct size, cpdf_toFileMemory (pdf, linearize,
 * make_id, &length) writes it and returns the buffer. The buffer length is
 * filled in &length. */
```

```
void *cpdf_toMemory (int, int, int, int *);

/* cpdf_isEncrypted(pdf) returns true if a documented is encrypted, false otherwise. */
int cpdf_isEncrypted (int);

/* Given a filename, is a PDF linearized? */
int is_linearized (const char[]);

/* cpdf_decryptPdf(pdf, userpw) attempts to decrypt a PDF using the given user
 * password. The error code is non-zero if the decryption fails. */
void cpdf_decryptPdf (int, const char[]);

/* cpdf_decryptPdfOwner(pdf, ownerpw) attempts to decrypt a PDF using the given
 * owner password. The error code is non-zero if the decryption fails. */
void cpdf_decryptPdfOwner (int, const char[]);

/* File permissions. These are inverted, in the sense that the presence of
 * one of them indicates a restriction. */
enum cpdf_permission
{
  cpdf_noEdit, /* Cannot edit the document */
  cpdf_noPrint, /* Cannot print the document */
  cpdf_noCopy, /* Cannot copy the document */
  cpdf_noAnnot, /* Cannot annotate the document */
  cpdf_noForms, /* Cannot edit forms in the document */
  cpdf_noExtract, /* Cannot extract information */
  cpdf_noAssemble, /* Cannot assemble into a bigger document */
  cpdf_noHqPrint /* Cannot print high quality */
};

/* Encryption methods. Suffixes 'false' and 'true' indicates lack of or
 * presence of encryption for XMP metadata streams. */
enum cpdf_encryptionMethod
{
  cpdf_pdf40bit, /* 40 bit RC4 encryption */
  cpdf_pdf128bit, /* 128 bit RC4 encryption */
  cpdf_aes128bitfalse, /* 128 bit AES encryption, do not encrypt metadata. */
  cpdf_aes128bittrue, /* 128 bit AES encryption, encrypt metadat */
  cpdf_aes256bitfalse, /* Deprecated. Do not use for new files */
  cpdf_aes256bittrue, /* Deprecated. Do not use for new files */
  cpdf_aes256bitisofalse, /* 256 bit AES encryption, do not encrypt metadata. */
  cpdf_aes256bitisotrue /* 256 bit AES encryption, encrypt metadata */
};

/* cpdf_toFileEncrypted(pdf, encryption_method, permissions, permission_length,
 * owner_password, user password, linearize, makeid) writes a file as
 * encrypted. */
void cpdf_toFileEncrypted
  (int, int, int *, int, const char[], const char[], int, int, const char[]);

/* cpdf_toFileEncryptedExt(pdf, encryption_method, permissions,
 * permission_length, owner_password, user_password, linearize, makeid,
 * preserve_objstm, generate_objstm, compress_objstm, filename) WARNING: the
 * pdf argument will be invalid after this call, and should be discarded. */
```

```
void cpdf_toFileEncryptedExt
  (int, int, int *, int, const char[], const char[], int, int, int, int, int,
   const char[]);

/* cpdf_hasPermission(pdf, permission) returns true if the given permission
 * (restriction) is present. */
int cpdf_hasPermission (int, enum cpdf_permission);

/* cpdf_encryptionMethod(pdf) return the encryption method currently in use on
 * a document. */
enum cpdf_encryptionMethod cpdf_encryptionKind (int);

/* Encryption and Permission status */

/* Internal status of a pdf loaded by the library. This is data kept separate
 * from the actual PDF. */
enum cpdf_pdfStatus
{
  cpdf_notEncrypted,
  cpdf_encrypted,
  cpdf_wasDecryptedWithUser,
  cpdf_wasDecryptedWithOwner
};

/* cpdf_lookupPdfStatus(pdf) returns the encryption status of a PDF. */
enum cpdf_pdfStatus cpdf_lookupPdfStatus (int);

/* cpdf_hasPermissionStatus(pdf, permission) returns true if the PDF has or had
 * the permission given set, assuming it is or was encrypted? */
int cpdf_hasPermissionStatus (int, enum cpdf_permission);

/* cpdf_encryptionMethod(pdf) returns what is (or was) the encryption method? */
enum cpdf_encryptionMethod cpdf_lookupPdfEncryption (int);

/* cpdf_lookupPdfUserPassword(pdf) finds the user password which was used to
 * decrypt a PDF, if is has status cpdf_wasDecryptedWithUser */
char *cpdf_lookupPdfUserPassword (int);
```

2 Merging and Splitting

3 Pages

4 Encryption

Covered in Chapter 1.

5 Compression

6 Bookmarks

7 Presentations

Not supported by libcpdf. Use the command line tools instead.

8 Logos, Watermarks and Stamps

9 Multipage Facilities

10 Annotations

Not supported in libcpdf. Use the command line tools instead.

11 Document Information and Metadata

12 File Attachments

13 Miscellaneous

14 Page Labels

A Dates

Dates in PDF are specified according to the following format:

D:YYYYMMDDHHmmSSOHH'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:2011 (2011)

D:20110103 (3rd March 2011)

D:201101031854-08'00' (3rd March 2011, 6:54PM, US Pacific Standard Time)

B Example Program in C

This program loads a file from disk and writes out a document with the original included three times. Note the use of `cpdf_startup`, `cpdf_lastError` and `cpdf_clearError`.

```
#include <stdbool.h>
#include "cpdflibwrapper.h"

int main (int argc, char ** argv)
{
  /* Initialise cpdf */
  cpdf_startup(argv);

  /* Clear the error state */
  cpdf_clearError();

  /* We will take the input hello.pdf and repeat it three times */
  int mergepdf = cpdf_fromFile("hello.pdf", "");

  /* Check the error state */
  if (cpdf_lastError) return 1;

  /* The array of PDFs to merge */
  int pdfs[] = {mergepdf, mergepdf, mergepdf};

  /* Clear the error state */
  cpdf_clearError();

  /* Merge them */
  int merged = cpdf_mergeSimple(pdfs, 3);

  /* Check the error state */
  if (cpdf_lastError) return 1;

  /* Clear the error state */
  cpdf_clearError();

  /* Write output */
  cpdf_toFile(merged, "merged.pdf", false, false);

  /* Check the error state */
  if (cpdf_lastError) return 1;

  return 0;
}
```