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

```
/* CHAPTER 2. Merging and Splitting */

/* cpdf_mergeSimple(pdfs, length) given an array of PDFs, and its length,
 * merges the files into a new one, which is returned. */
int cpdf_mergeSimple (int *, int);

/* cpdf_merge(pdfs, len, retain_numbering, remove_duplicate_fonts) 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. */
int cpdf_merge (int *, int, int, int);

/* cpdf_mergeSame(pdfs, len, retain_numbering, remove_duplicate_fonts, ranges)
 * is the same as cpdf_merge, except that it has an additional argument
 * - an array 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. */
int cpdf_mergeSame (int *, int, int, int, int *);

/* cpdf_selectPages(pdf, range) returns a new document which just those pages
 * in the page range. */
int cpdf_selectPages (int, int);
```

3 Pages

```
/* CHAPTER 3. Pages */

/* cpdf_scalePages(pdf, range, x scale, y scale) scales the page dimensions
 * and content by the given scale, about (0, 0). Other boxes (crop etc. are
 * altered as appropriate) */
void cpdf_scalePages (int, int, double, double);

/* cpdf_scaleToFit(pdf, range, width height, scale) scales the content to fit
 * new page dimensions (width x height) multiplied by scale (typically 1.0).
 * Other boxed (crop etc. are altered as appropriate) */
void cpdf_scaleToFit (int, int, double, double, double);

/* cpdf_scaleToFitPaper(pdf, range, papersize, scale) scales the page content
 * to fit the given page size, possibly multiplied by scale (typically 1.0) */
void cpdf_scaleToFitPaper (int, int, enum cpdf_papersize, double);

/* Positions on the page. Used for scaling about a point, and adding text. */
enum cpdf_anchor
{
  cpdf_posCentre, /* Absolute centre */
  cpdf_posLeft, /* Absolute left */
  cpdf_posRight, /* Absolute right */
  cpdf_top, /* Top top centre of the page */
  cpdf_topLeft, /* The top left of the page */
  cpdf_topRight, /* The top right of the page */
  cpdf_left, /* The left hand side of the page, halfway down */
  cpdf_bottomLeft, /* The bottom left of the page */
  cpdf_bottom, /* The bottom middle of the page */
  cpdf_bottomRight, /* The bottom right of the page */
  cpdf_right, /* The right hand side of the page, halfway down */
  cpdf_diagonal, /* Diagonal, bottom left to top right */
  cpdf_reverseDiagonal /* Diagonal, top left to bottom right */
};

/* A cpdf_position is an anchor (above) and zero or one or two parameters
 * (cpdf_coord1, cpdf_coord2).
 *
 * cpdf_posCentre: Two parameters, x and y
 * cpdf_posLeft: Two parameters, x and y
 * cpdf_posRight: Two parameters, x and y
 * cpdf_top: One parameter -- distance from top
 * cpdf_topLeft: One parameter -- distance from top left
```

```
 * cpdf_topRight: One parameter -- distance from top right
 * cpdf_left: One parameter -- distance from left middle
 * cpdf_bottomLeft: One parameter -- distance from bottom left
 * cpdf_bottom: One parameter -- distance from bottom
 * cpdf_bottomRight: One parameter -- distance from bottom right
 * cpdf_right: One parameter -- distance from right
 * cpdf_diagonal: Zero parameters
 * cpdf_reverseDiagonal: Zero paremeters */
struct cpdf_position
{
 int cpdf_anchor; /* Position anchor */
 double cpdf_coord1; /* Parameter one */
 double cpdf_coord2; /* Parameter two */
};

/* cpdf_scaleContents(pdf, range, position, scale) scales the contents of the
 * pages in the range about the point given by the cpdf_position, by the
 * scale given. */
void cpdf_scaleContents (int, int, struct cpdf_position, double);

/* cpdf_shiftContents(pdf, range, dx, dy) shifts the content of the pages in
 * the range. */
void cpdf_shiftContents (int, int, double, double);

/* cpdf_rotate(pdf, range, rotation) changes the viewing rotation to an
 * absolute value. Appropriate rotations are 0, 90, 180, 270. */
void cpdf_rotate (int, int, int);

/* cpdf_rotateBy(pdf, range, rotation) changes the viewing rotation by a
 * given number of degrees. Appropriate values are 90, 180, 270. */
void cpdf_rotateBy (int, int, int);

/* cpdf_rotateContents(pdf, range, angle) rotates the content about the centre
 * of the page by the given number of degrees, in a clockwise direction. */
void cpdf_rotateContents (int, int, double);

/* cpdf_upright(pdf, range) changes the viewing rotation of the pages in the
 * range, counter-rotating the dimensions and content such that there is no
 * visual change. */
void cpdf_upright (int, int);

/* cpdf_hFlip(pdf, range) flips horizontally the pages in the range. */
void cpdf_hFlip (int, int);

/* cpdf_vFlip(pdf, range) flips vertically the pages in the range. */
void cpdf_vFlip (int, int);

/* cpdf_crop(pdf, range, x, y, w, h) crops a page, replacing any existing
 * crop box. The dimensions are in points. */
void cpdf_crop (int, int, double, double, double, double);

/* cpdf_removeCrop(pdf, range) removes any crop box from pages in the range. */
void cpdf_removeCrop (int, int);
```

```
/* cpdf_removeTrim(pdf, range) removes any crop box from pages in the range. */
void cpdf_removeTrim (int, int);

/* cpdf_removeArt(pdf, range) removes any crop box from pages in the range. */
void cpdf_removeArt (int, int);

/* cpdf_removeBleed(pdf, range) removes any crop box from pages in the range. */
void cpdf_removeBleed (int, int);

/* cpdf_trimMarks(pdf, range) adds trim marks to the given pages, if the trimbox exists. */
void cpdf_trimMarks (int, int);

/* cpdf_showBoxes(pdf, range) shows the boxes on the given pages, for debug. */
void cpdf_showBoxes (int, int);

/* cpdf_hardBox make a given box a 'hard box' i.e clips it explicitly. */
void cpdf_hardBox (int, int, const char[]);
```

4 Encryption

Covered in Chapter 1.

```
/* CHAPTER 4. Encryption */

/* Encryption covered under Chapter 1 in cpdflib. */
```

5 Compression

```
/* CHAPTER 5. Compression */

/* cpdf_compress(pdf) compresses any uncompressed streams in the given PDF
 * using the Flate algorithm. */
void cpdf_compress (int);

/* cpdf_uncompress(pdf) uncompresses any streams in the given PDF, so long as
 * the compression method is supported. */
void cpdf_decompress (int);

/* cpdf_squeeze(userpw, logfile, infile, outfile) squeezes a file on disk,
 * logging to logfile if given. */
int cpdf_squeeze (const char[], const char[], const char[], const char[]);

/* cpdf_squeezeToMemory(pdf) squeezes a pdf in memory. */
void cpdf_squeezeInMemory (int);
```

6 Bookmarks

```
/* CHAPTER 6. Bookmarks */

/* cpdf_startGetBookmarkInfo(pdf) start the bookmark retrieval process for a
 * given PDF. */
void cpdf_startGetBookmarkInfo (int);

/* cpdf_numberBookmarks gets the number of bookmarks for the PDF given to
 * cpdf_startGetBookmarkInfo. */
int cpdf_numberBookmarks (void);

/* cpdf_getBookmarkLevel(serial) get bookmark level for the given bookmark
 * (0...(n - 1)). */
int cpdf_getBookmarkLevel (int);

/* cpdf_getBookmarkPage gets the bookmark target page for the given PDF (which
 * must be the same as the PDF passed to cpdf_startSetBookmarkInfo) and
 * bookmark (0...(n - 1)). */
int cpdf_getBookmarkPage (int, int);

/* cpdf_getBookmarkText returns the text of bookmark (0...(n - 1)). */
char *cpdf_getBookmarkText (int);

/* cpdf_getBookmarkOpenStatus(pdf) is true if the bookmark is open. */
int cpdf_getBookmarkOpenStatus (int);

/* cpdf_endGetBookmarkInfo ends the bookmark retrieval process, cleaning up. */
void cpdf_endGetBookmarkInfo (void);

/* cpdf_startGetBookmarkInfo(n) start the bookmark setting process for n
 * bookmarks. */
void cpdf_startSetBookmarkInfo (int);

/* cpdf_setBookmarkLevel(n, level) set bookmark level for the given bookmark
 * (0...(n - 1)). */
void cpdf_setBookmarkLevel (int, int);

/* cpdf_setBookmarkPage(pdf, bookmark, targetpage) sets the bookmark target
 * page for the given PDF (which must be the same as the PDF to be passed to
 * cpdf_endSetBookmarkInfo) and bookmark (0...(n - 1)). */
void cpdf_setBookmarkPage (int, int, int);

/* cpdf_setBookmarkOpenStatus(n, status) set the open status of a bookmark,
```

```
 * true or false. */
void cpdf_setBookmarkOpenStatus (int, int);

/* cpdf_setBookmarkText(n, text) sets the text of bookmark (0...(n - 1)). */
void cpdf_setBookmarkText (int, const char[]);

/* cpdf_endSetBookmarkInfo(pdf) end the bookmark setting process, writing the
 * bookmarks to the given PDF. */
void cpdf_endSetBookmarkInfo (int);
```

7 Presentations

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

```
/* CHAPTER 7. Presentations */

/* Not included in the library version */
```

8 Logos, Watermarks and Stamps

```
/* CHAPTER 8. Logos, Watermarks and Stamps */

/* cpdf_stampOn(stamp_pdf, pdf, range) 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. */
void cpdf_stampOn (int, int, int);

/* cpdf_stampUnder(stamp_pdf, pdf, range) 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. */
void cpdf_stampUnder (int, int, int);

/* cpdf_stampExtended(pdf, pdf2, range, isover, scale_stamp_to_fit, pos,
 * relative_to_cropbox) is 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_cropbox: if true, pos is relative to cropbox not mediabox */
void cpdf_stampExtended (int, int, int, int, int, struct cpdf_position, int);

/* cpdf_combinePages(under, over) combines the PDFs page-by-page, putting
 * each page of 'over' over each page of 'under' */
int cpdf_combinePages (int, int);

/* Adding text. Adds text to a PDF, if the characters exist in the font. */

/* Special codes
 *
 * %Page Page number in arabic notation (1, 2, 3...)
 * %roman Page number in lower-case roman notation (i, ii, iii...)
 * %Roman Page number in upper-case roman notation (I, II, III...)
 * %EndPage Last page of document in arabic notation
 * %Label The page label of the page
 * %EndLabel The page label of the last page
 * %filename The full file name of the input document
 * %a Abbreviated weekday name (Sun, Mon etc.)
 * %A Full weekday name (Sunday, Monday etc.)
 * %b Abbreviated month name (Jan, Feb etc.)
 * %B Full month name (January, February etc.)
 * %d Day of the month (01-31)
 * %e Day of the month (1-31)
 * %H Hour in 24-hour clock (00-23)
```

```
 * %I Hour in 12-hour clock (01-12)
 * %j Day of the year (001-366)
 * %m Month of the year (01-12)
 * %M Minute of the hour (00-59)
 * %p "a.m" or "p.m"
 * %S Second of the minute (00-61)
 * %T Same as %H:%M:%S
 * %u Weekday (1-7, 1 = Monday)
 * %w Weekday (0-6, 0 = Monday)
 * %Y Year (0000-9999)
 * %% The % character */

/* The standard fonts */
enum cpdf_font
{
 cpdf_timesRoman, /* Times Roman */
 cpdf_timesBold, /* Times Bold */
 cpdf_timesItalic, /* Times Italic */
 cpdf_timesBoldItalic, /* Times Bold Italic */
 cpdf_helvetica, /* Helvetica */
 cpdf_helveticaBold, /* Helvetica Bold */
 cpdf_helveticaOblique, /* Helvetica Oblique */
 cpdf_helveticaBoldOblique, /* Helvetica Bold Oblique */
 cpdf_courier, /* Courier */
 cpdf_courierBold, /* Courier Bold */
 cpdf_courierOblique, /* Courier Oblique */
 cpdf_courierBoldOblique /* Courier Bold Oblique */
};

/* Justifications for multi line text */
enum cpdf_justification
{
 cpdf_leftJustify, /* Left justify */
 cpdf_CentreJustify, /* Centre justify */
 cpdf_RightJustify /* Right justify */
};

/* Add text */
void cpdf_addText (int, /* If true, don't actually add text but collect metrics. *
               int, /* Document */
               int, /* Page Range */
               const char[], /* The text to add */
               struct cpdf_position, /* Position to add text at */
               double, /* Linespacing, 1.0 = normal */
               int, /* Starting Bates number */
               enum cpdf_font, /* Font */
               double, /* Font size in points */
               double, /* Red component of colour, 0.0 - 1.0 */
               double, /* Green component of colour, 0.0 - 1.0 */
               double, /* Blue component of colour, 0.0 - 1.0 */
               int, /* If true, text is added underneath rather than on top */
               int, /* If true, position is relative to crop box not media box */
               int, /* If true, text is outline rather than filled */
               double, /* Opacity, 1.0 = opaque, 0.0 = wholly transparent */
```

```
                enum cpdf_justification, /* Justification */
                int, /* If true, position is relative to midline of text, not baseline */
                int, /* If true, position is relative to topline of text, not baseline */
                const char[], /* filename that this document was read from (optional) */
                double, /* line width */
                int /* embed fonts */
  );

/* Add text, with most parameters default */
void cpdf_addTextSimple (int, /* Document */
                         int, /* Page range */
                         const char[], /* The text to add */
                         struct cpdf_position, /* Position to add text at */
                         enum cpdf_font, /* font */
                         double); /* font size */

/* To return metrics about the text which would be added. Call cpdf_addText
 * first with the first argument set to false, and the other arguments filled
 * in as appropriate. Now, the metrics have been collected. Call
 * cpdf_addTextHowMany to find out how many lines of text there are. Now, for
 * each line (0...n-1), the functions cpdf_addTextReturn* give the metrics of
 * the text as calculated. */
int cpdf_addTextHowMany (void);
char *cpdf_addTextReturnText (int);
double cpdf_addTextReturnX (int);
double cpdf_addTextReturnY (int);
double cpdf_addTextReturnRotation (int);
double cpdf_addTextReturnBaselineAdjustment (void);

/* cpdf_removeText(pdf, range) will remove any text added by libcpdf from the
 * given pages. */
void cpdf_removeText (int, int);

/* Return the width of a given string in the given font in thousandths of a
 * point. */
int cpdf_textWidth (enum cpdf_font, const char[]);
```

9 Multipage Facilities

```
/* CHAPTER 9. Multipage facilities */

/* Impose a document two up. cpdf_twoUp does so by retaining the existing page
 * size, scaling pages down. cpdf_twoUpStack does so by doubling the page size,
 * to fit two pages on one. */
void cpdf_twoUp (int);
void cpdf_twoUpStack (int);

/* cpdf_padBefore(pdf, range) adds a blank page before each page in the given range */
void cpdf_padBefore (int, int);

/* cpdf_padBefore(pdf, range) adds a blank page before each page in the given range */
void cpdf_padAfter (int, int);

/* cpdf_pageEvery(pdf, n) adds a blank page after every n pages */
void cpdf_padEvery (int, int);

/* cpdf_padMultiple(pdf, n) adds pages at the end to pad the file to a multiple
 * of n pages in length. */
void cpdf_padMultiple (int, int);

/* cpdf_padMultiple(pdf, n) adds pages at the beginning to pad the file to a
 * multiple of n pages in length. */
void cpdf_padMultipleBefore (int, int);
```

10 Annotations

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

```
/* CHAPTER 10. Annotations */

/* Not in the library version */
```

11 Document Information and Metadata

```
/* CHAPTER 11. Document Information and Metadata */

/* cpdf_isLinearized(filename) finds out if a document is linearized as quickly
 * as possible without loading it. */
int cpdf_isLinearized (const char[]);

/* cpdf_vetVersion(pdf) returns the minor version number of a document. */
int cpdf_getVersion (int);

/* cpdf_vetMajorVersion(pdf) returns the minor version number of a document. */
int cpdf_getMajorVersion (int);

/* cpdf_getTitle(pdf) returns the title of a document. */
char *cpdf_getTitle (int);

/* cpdf_getAuthor(pdf) returns the author of a document. */
char *cpdf_getAuthor (int);

/* cpdf_getSubject(pdf) returns the subject of a document. */
char *cpdf_getSubject (int);

/* cpdf_getKeywords(pdf) returns the keywords of a document. */
char *cpdf_getKeywords (int);

/* cpdf_getCreator(pdf) returns the creator of a document. */
char *cpdf_getCreator (int);

/* cpdf_getProducer(pdf) returns the producer of a document. */
char *cpdf_getProducer (int);

/* cpdf_getCreationDate(pdf) returns the creation date of a document. */
char *cpdf_getCreationDate (int);

/* cpdf_getModificationDate(pdf) returns the modification date of a document. */
char *cpdf_getModificationDate (int);

/* cpdf_getTitleXMP(pdf) returns the XMP title of a document. */
char *cpdf_getTitleXMP (int);

/* cpdf_getAuthorXMP(pdf) returns the XMP author of a document. */
char *cpdf_getAuthorXMP (int);
```

```
/* cpdf_getSubjectXMP(pdf) returns the XMP subject of a document. */
char *cpdf_getSubjectXMP (int);

/* cpdf_getKeywordsXMP(pdf) returns the XMP keywords of a document. */
char *cpdf_getKeywordsXMP (int);

/* cpdf_getCreatorXMP(pdf) returns the XMP creator of a document. */
char *cpdf_getCreatorXMP (int);

/* cpdf_getProducerXMP(pdf) returns the XMP producer of a document. */
char *cpdf_getProducerXMP (int);

/* cpdf_getCreationDateXMP(pdf) returns the XMP creation date of a document. */
char *cpdf_getCreationDateXMP (int);

/* cpdf_getModificationDateXMP(pdf) returns the XMP modification date of a documen
char *cpdf_getModificationDateXMP (int);

/* cpdf_setTitle(pdf) sets the title of a document. */
void cpdf_setTitle (int, const char[]);

/* cpdf_setAuthor(pdf) sets the author of a document. */
void cpdf_setAuthor (int, const char[]);

/* cpdf_setSubject(pdf) sets the subject of a document. */
void cpdf_setSubject (int, const char[]);

/* cpdf_setKeywords(pdf) sets the keywords of a document. */
void cpdf_setKeywords (int, const char[]);

/* cpdf_setCreator(pdf) sets the creator of a document. */
void cpdf_setCreator (int, const char[]);

/* cpdf_setProducer(pdf) sets the producer of a document. */
void cpdf_setProducer (int, const char[]);

/* cpdf_setCreationDate(pdf) sets the creation date of a document. */
void cpdf_setCreationDate (int, const char[]);

/* cpdf_setModificationDate(pdf) sets the modifcation date of a document. */
void cpdf_setModificationDate (int, const char[]);

/* cpdf_setTitleXMP(pdf) set the XMP title of a document. */
void cpdf_setTitleXMP (int, const char[]);

/* cpdf_setAuthorXMP(pdf) set the XMP author of a document. */
void cpdf_setAuthorXMP (int, const char[]);

/* cpdf_setSubjectXMP(pdf) set the XMP subject of a document. */
void cpdf_setSubjectXMP (int, const char[]);

/* cpdf_setKeywordsXMP(pdf) set the XMP keywords of a document. */
void cpdf_setKeywordsXMP (int, const char[]);
```

```
/* cpdf_setCreatorXMP(pdf) set the XMP creator of a document. */
void cpdf_setCreatorXMP (int, const char[]);

/* cpdf_setProducerXMP(pdf) set the XMP producer of a document. */
void cpdf_setProducerXMP (int, const char[]);

/* cpdf_setCreationDateXMP(pdf) set the XMP creation date of a document. */
void cpdf_setCreationDateXMP (int, const char[]);

/* cpdf_setModificationDateXMP(pdf) set the XMP modification date of a document. */
void cpdf_setModificationDateXMP (int, const char[]);

/* Dates: Month 1-31, day 1-31, hours (0-23), minutes (0-59), seconds (0-59),
 * hour_offset is the offset from UT in hours (-23 to 23); minute_offset is the
 * offset from UT in minutes (-59 to 59). */

/* cpdf_getDateComponents(datestring, year, month, day, hour, minute, second,
 * hour_offset, minute_offset) returns the components from a PDF date string. */
void cpdf_getDateComponents (const char[], int *, int *, int *, int *, int *,
                     int *, int *, int *);

/* cpdf_dateStringOfComponents(year, month, day, hour, minute, second,
 * hour_offset, minute_offset) builds a PDF date string from individual
 * components. */
char *cpdf_dateStringOfComponents (int, int, int, int, int, int, int, int);

/* cpdf_getPageRotation(pdf, pagenumber) gets the viewing rotation for a given
 * page. */
int cpdf_getPageRotation (int, int);

/* cpdf_hasBox(pdf, pagenumber, boxname) returns true, if that page has the
 * given box. E.g "/CropBox" */
int cpdf_hasBox (int, int, const char[]);

/* These functions get a box given the document, page range, min x, max x, min y, max y in
 * points. Only suceeds if such a box exists, as checked by cpdf_hasBox */
void cpdf_getMediaBox (int, int, double *, double *, double *, double *);
void cpdf_getCropBox (int, int, double *, double *, double *, double *);
void cpdf_getTrimBox (int, int, double *, double *, double *, double *);
void cpdf_getArtBox (int, int, double *, double *, double *, double *);
void cpdf_getBleedBox (int, int, double *, double *, double *, double *);

/* These functions set a box given the document, page range, min x, max x, min
 * y, max y in points. */
void cpdf_setMediabox (int, int, double, double, double, double);
void cpdf_setCropBox (int, int, double, double, double, double);
void cpdf_setTrimBox (int, int, double, double, double, double);
void cpdf_setArtBox (int, int, double, double, double, double);
void cpdf_setBleedBox (int, int, double, double, double, double);

/* cpdf_markTrapped(pdf) marks a document as trapped. */
void cpdf_markTrapped (int);

/* cpdf_markUntrapped(pdf) marks a document as untrapped. */
```

```
void cpdf_markUntrapped (int);

/* cpdf_markTrappedXMP(pdf) marks a document as trapped in XMP metadata. */
void cpdf_markTrappedXMP (int);

/* cpdf_markUntrappedXMP(pdf) marks a document as untrapped in XMP metadata. */
void cpdf_markUntrappedXMP (int);

/* Document Layouts. */
enum cpdf_layout
{
 cpdf_singlePage,
 cpdf_oneColumn,
 cpdf_twoColumnLeft,
 cpdf_twoColumnRight,
 cpdf_twoPageLeft,
 cpdf_twoPageRight
};

/* cpdf_setPageLayout(pdf, layout) sets the page layout for a document. */
void cpdf_setPageLayout (int, enum cpdf_layout);

/* Document page modes. */
enum cpdf_pageMode
{
 cpdf_useNone,
 cpdf_useOutlines,
 cpdf_useThumbs,
 cpdf_useOC,
 cpdf_useAttachments
};

/* cpdf_setPageMode(pdf, mode) sets the page mode for a document. */
void cpdf_setPageMode (int, enum cpdf_pageMode);

/* cpdf_hideToolbar(pdf, flag) sets the hide toolbar flag */
void cpdf_hideToolbar (int, int);

/* cpdf_hideMenubar(pdf, flag) sets the hide menu bar flag */
void cpdf_hideMenubar (int, int);

/* cpdf_hideWindowUi(pdf, flag) sets the hide window UI flag */
void cpdf_hideWindowUi (int, int);

/* cpdf_fitWindow(pdf, flag) sets the fit window flag */
void cpdf_fitWindow (int, int);

/* cpdf_centerWindow(pdf, flag) sets the center window flag */
void cpdf_centerWindow (int, int);

/* cpdf_displayDocTitle(pdf, flag) sets the display doc title flag */
void cpdf_displayDocTitle (int, int);

/* cpdf_openAtPage(pdf, fit, pagenumber) */
```

```
void cpdf_openAtPage (int, int, int);

/* cpdf_setMetadataFromFile(pdf, filename) set the XMP metadata of a document,
 * given a file name. */
void cpdf_setMetadataFromFile (int, const char[]);

/* cpdf_setMetadataFromByteArray(pdf, data, length) set the XMP metadata from
 * an array of bytes. */
void cpdf_setMetadataFromByteArray (int, void *, int);

/* cpdf_getMetadata(pdf, &length) returns the XMP metadata and fills in length. */
void *cpdf_getMetadata (int, int *);

/* cpdf_removeMetadata(pdf) removes the XMP metadata from a document */
void cpdf_removeMetadata (int);

/* cpdf_createMetadata(pdf) builds fresh metadata as best it can from existing
 * metadata in the document. */
void cpdf_createMetadata (int);

/* cpdf_setMetadataDate(pdf, date) 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. */
void cpdf_setMetadataDate (int, const char[]);

/* Styles of page label */
enum cpdf_pageLabelStyle
{
 cpdf_decimalArabic, /* 1,2,3... */
 cpdf_uppercaseRoman, /* I, II, III... */
 cpdf_lowercaseRoman, /* i, ii, iii... */
 cpdf_uppercaseLetters, /* A, B, C... */
 cpdf_lowercaseLetters /* a, b, c... */
};

/* Add a page labels.
 *
 * cpdf_addPageLabels(pdf, style, prefix, offset, range, progress)
 *
 * 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. */
void cpdf_addPageLabels (int, enum cpdf_pageLabelStyle, const char[], int,
                  int, int);

/* cpdf_removePageLabels(pdf) removes the page labels from the document. */
void cpdf_removePageLabels (int);

/* cpdf_getPageLabelStringForPage(pdf, page number) calculates the full label
 * string for a given page, and returns it */
char *cpdf_getPageLabelStringForPage (int, int);

/* Get page label data. Call cpdf_startGetPageLabels to find out how many there
 * are, then use these serial numbers to get the style, prefix, offset and
 * range. Call cpdf_endGetPageLabels to clean up. */
```

```
int cpdf_startGetPageLabels (int);
enum cpdf_pageLabelStyle cpdf_getPageLabelStyle (int);
char *cpdf_getPageLabelPrefix (int);
int cpdf_getPageLabelOffset (int);
int cpdf_getPageLabelRange (int);
void cpdf_endGetPageLabels ();
```

12 File Attachments

```
/* CHAPTER 12. File Attachments */

/* cpdf_attachFile(filename, pdf) attaches a file to the pdf. It is attached at
 * document level. */
void cpdf_attachFile (const char[], int);

/* cpdf_attachFileToPage(filename, pdf, pagenumber) attaches a file, given its
 * file name, pdf, and the page number to which it should be attached. */
void cpdf_attachFileToPage (const char[], int, int);

/* cpdf_attachFileFromMemory(memory, length, filename, pdf) attaches from
 * memory, just like cpdf_attachFile. */
void cpdf_attachFileFromMemory (void *, int, const char[], int);

/* cpdf_attachFileToPageFromMemory(memory, length, filename, pdf, pagenumber)
 * attaches from memory, just like cpdf_attachFileToPage. */
void cpdf_attachFileToPageFromMemory (void *, int, const char[], int, int);

/* Remove all page- and document-level attachments from a document */
void cpdf_removeAttachedFiles (int);

/* List information about attachments. Call cpdf_startGetAttachments(pdf) first,
 * then cpdf_numberGetAttachments to find out how many there are. Then
 * cpdf_getAttachmentName to return each one 0...(n - 1). Finally, call
 * cpdf_endGetAttachments to clean up. */
void cpdf_startGetAttachments (int);

/* Get the number of attachments. */
int cpdf_numberGetAttachments (void);

/* Get the name of the attachment. */
char *cpdf_getAttachmentName (int);

/* Gets the page number. 0 = document level */
int cpdf_getAttachmentPage (int);

/* cpdf_getAttachmentData(serial number, &length) returns a pointer to the
 * data, and its length */
void *cpdf_getAttachmentData (int, int *);

/* Clean up. */
void cpdf_endGetAttachments (void);
```

13 Images

```
/* CHAPTER 13. Images. */

/* Get image data, including resolution at all points of use. Call
 * cpdf_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..<total number> - 1, to retrieve the data. Finally, call
 * cpdf_endGetImageResolution to clean up. */
int cpdf_startGetImageResolution (int, int);
int cpdf_getImageResolutionPageNumber (int);
char *cpdf_getImageResolutionImageName (int);
int cpdf_getImageResolutionXPixels (int);
int cpdf_getImageResolutionYPixels (int);
double cpdf_getImageResolutionXRes (int);
double cpdf_getImageResolutionYRes (int);
void cpdf_endGetImageResolution (void);
```

14 Fonts

```
/* CHAPTER 14. Fonts. */

/* Retrieving font information. First, call cpdf_startGetFontInfo(pdf). Now
 * call cpdf_numberFonts to return the number of fonts. For each font, call one
 * or more of cpdf_getFontPage, cpdf_getFontName, cpdf_getFontType, and
 * cpdf_getFontEncoding giving a serial number 0..<number of fonts> - 1 to return
 * information. Finally, call cpdf_endGetFontInfo to clean up. */
void cpdf_startGetFontInfo (int);
int cpdf_numberFonts (void);
int cpdf_getFontPage (int);
char *cpdf_getFontName (int);
char *cpdf_getFontType (int);
char *cpdf_getFontEncoding (int);
void cpdf_endGetFontInfo (void);

/* cpdf_removeFonts(pdf) removes all font data from a file. */
void cpdf_removeFonts (int);

/* cpdf_copyFont(from, to, range, pagenumber, fontname) 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 it's font name. */
void cpdf_copyFont (int, int, int, int, const char[]);
```

15 Miscellaneous

```
/* CHAPTER 15. Miscellaneous */

/* cpdf_draft(pdf, range, boxes) removes images on the given pages, replacing
 * them with crossed boxes if 'boxes' is true */
void cpdf_draft (int, int, int);

/* cpdf_removeAllText(pdf, range) removes all text from the given pages in a
 * given document. */
void cpdf_removeAllText (int, int);

/* cpdf_blackText(pdf, range) blackens all text on the given pages. */
void cpdf_blackText (int, int);

/* cpdf_blackLines(pdf, range) blackens all lines on the given pages. */
void cpdf_blackLines (int, int);

/* cpdf_blackFills(pdf, range) blackens all fills on the given pages. */
void cpdf_blackFills (int, int);

/* cpdf_thinLines(pdf, range, min_thickness) thickens every line less than
 * min_thickness to min_thickness. Thickness given in points. */
void cpdf_thinLines (int, int, double);

/* cpdf_copyId(from, to) copies the /ID from one document to another. */
void cpdf_copyId (int, int);

/* cpdf_removeId(pdf) removes a document's /ID */
void cpdf_removeId (int);

/* cpdf_setVersion(pdf, version) sets the minor version number of a document. */
void cpdf_setVersion (int, int);

/* cpdf_removeDictEntry(pdf, key) removes any dictionary entry with the given
 * key anywhere in the document */
void cpdf_removeDictEntry (int, const char[]);

/* cpdf_removeClipping(pdf, range) removes all clipping from pages in the given
 * range */
void cpdf_removeClipping (int, int);

/* CHAPTER UNDOC */
```

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;
}
```