

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

©2017 Coherent Graphics Limited. All rights reserved.

Adobe, Acrobat, Adobe PDF, Adobe Reader and PostScript are registered trademarks of Adobe Systems Incorporated. Windows is a registered trademarks of Microsoft Corporation.

Contents

Contents	iii
0 Preliminaries	5
1 Basics	7
2 Merging and Splitting	13
3 Pages	15
4 Encryption	19
5 Compression	21
6 Bookmarks	23
7 Presentations	25
8 Logos, Watermarks and Stamps	27
9 Multipage Facilities	31
10 Annotations	33
11 Document Information and Metadata	35
12 File Attachments	39
13 Miscellaneous	41
14 Page Labels	43
15 Special functionality 1 – Encryption and Permission status	45
16 Special functionality 2 – Undo	47
A Dates	49
B Example Program in C	51

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 cpdfplib 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 `cpdfplibwrapper.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 `cpdfplibwrapper.h` somewhere suitable and instruct 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 `cpdfplibwrapper.h` somewhere suitable and instruct your C compiler to search for headers there. The library is now ready for use.

0 Preliminaries

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

/* Set demo mode. Upon library startup is false. If set, files written will
 * have the text DEMO stamped over each page. This stamping will also slow down
 * the library significantly. */
void cpdf_setDemo(int);

/* Errors. lastError and 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. */
int cpdf_lastError;
char* cpdf_lastErrorString;

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

/* 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);

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

/* Calling 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);
```

1 Basics

```
/* 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 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 is the last page, the penultimate page
etc. */

/* Parse a page specification with reference to a given PDF (the PDF is
```

1. BASICS

```
 * supplied so that page ranges which reference pages which do not exist are
 * rejected). */
int cpdf_parsePagespec(int, char*);

/* Validates a page specification so far as is possible in the absence of the
 * actual document */
int cpdf_validatePagespec(char *);

/* Build 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);

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

/* Delete a range. Ranges should be deleted once finished with. */
void cpdf_deleteRange(int);

/* 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);

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

/* Make a range which contains just the odd pages of another range */
int cpdf_odd(int);

/* Make the union of two ranges cpdf_union(a,b) gives a range containing the
 * pages in range a and range b. */
int cpdf_rangeUnion(int, int);

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

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

/* Give the number of pages in a range */
int cpdf_rangeLength(int);

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

/* Calling cpdf_rangeAdd(range, page) will add 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);
```

```

/* Load a PDF file from a given file. 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. */
int cpdf_fromFile(char*, char*);

/* Load 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. */
int cpdf_fromFileLazy(char*, char*);

/* Create 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 */};

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

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

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

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

```

1. BASICS

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

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

/* Attempt to decrypt a PDF using the given owner password. The error code is
 * non-zero if the decryption fails. */
void cpdf_decryptPdfOwner(int, 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 XML 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_aes256bitisottrue /** 256 bit AES encryption, encrypt metadata */};

/* Write a file with encryption:
cpdf_toFileEncrypted(
    pdf, Document
    encryption_method, Encryption method
    permissions, Permissions array
    permission_length, Length of permissions array
    owner_password, Owner password, blank if none
    user_passwordi, User password, blank if none
    linearize, If true, linearize
    makeid, If true, make a new ID
    filename) Filename
*/
void cpdf_toFileEncrypted(int, int, int*, int, char*, char*, int, int, char*);

/* Write a modified file, re-encrypting it.

cpdf_toFileReencrypting(original, decrypted_and_modified, userpw, filename)
```

original is the original document, as read from file. decrypted_and_modified is the processed file, ready to write. userpw is the user password for the PDF. filename is the name of the file to write.

The PDF 'modified' is no longer usable and can be deleted.

**/*

void cpdf_toFileRecrypting(**int**, **int**, **char***, **char***);

/ Return true if the given permission (restriction) is present. */*

int cpdf_hasPermission(**int**, **enum** cpdf_permission);

/ Return the encryption method currently in use on a file. */*

enum cpdf_encryptionMethod cpdf_encryptionKind(**int**);

2 Merging and Splitting

```
/* Given an array of PDFs, and its length, merge the files into a new one */
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);

/* This 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

```
/* 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) scales the content to fit new page
 * dimensions (width x height). Other boxed (crop etc. are altered as
 * appropriate) */
void cpdf_scaleToFit(int, int, double, double);

/* cpdf_scaleToFitPaper(pdf, range, papersize) scales the page content to fit
 * the given page size. */
void cpdf_scaleToFitPaper(int, int, enum cpdf_papersize);

/* 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 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
```

3. PAGES

```
* 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 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);
```

4 Encryption

Covered in Chapter 1.

5 Compression

```
/* These functions can be used to compress and decompress all the streams in a
* PDF file, for example for manual inspection. A PDF's streams are typically
* compressed. Do not expect compression to reduce the size of an
* already-compressed PDF. */

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

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

6 Bookmarks

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

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

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

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

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

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

7 Presentations

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

8 Logos, Watermarks and Stamps

```
/* cpdf_stampOn(pdf, stamp_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_stampOn(pdf, stamp_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_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 UTF8 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.
```

8. LOGOS, WATERMARKS AND STAMPS

```
*/

/** 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 */,
 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 */,
 char* /** filename that this document was read from (optional) */
);

/* 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 (1...n), 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 will remove any text added by libcpdf. */  
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, char);
```

9 Multipage Facilities

```
/* Impose a document two up */
void cpdf_twoUp(int);

/* Pad a document (first argument) before each page in the given range (second
* argument) */
void cpdf_padBefore(int, int);

/* Pad a document (first argument) after each page in the given range (second
* argument) */
void cpdf_padAfter(int, int);
```

10 Annotations

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

11 Document Information and Metadata

```
/* 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 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);  
  
/* Find out if a document is linearized as quickly as possible without loading  
 * it. */  
int cpdf_isLinearized(char*);  
  
/* Return the minor version number of a document. */  
int cpdf_getVersion(int);  
  
/* Return the title of a document. */  
char* cpdf_getTitle(int);  
  
/* Return the author of a document. */  
char* cpdf_getAuthor(int);  
  
/* Return the subject of a document. */  
char* cpdf_getSubject(int);  
  
/* Return the keywords of a document. */  
char* cpdf_getKeywords(int);  
  
/* Return the creator of a document. */  
char* cpdf_getCreator(int);  
  
/* Return the producer of a document. */  
char* cpdf_getProducer(int);  
  
/* Return the creation date of a document. */  
char* cpdf_getCreationDate(int);  
  
/* Return the modification date of a document. */
```

11. DOCUMENT INFORMATION AND METADATA

```
char* cpdf_getModificationDate(int);

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

/* Set the title of a document from a UTF8 encoded string */
void cpdf_setTitle(int, char*);

/* Set the author of a document from a UTF8 encoded string */
void cpdf_setAuthor(int, char*);

/* Set the subject of a document from a UTF8 encoded string */
void cpdf_setSubject(int, char*);

/* Set the keywords of a document from a UTF8 encoded string */
void cpdf_setKeywords(int, char*);

/* Set the creator of a document from a UTF8 encoded string */
void cpdf_setCreator(int, char*);

/* Set the producer of a document from a UTF8 encoded string */
void cpdf_setProducer(int, char*);

/* Set the creation date of a document from a UTF8 encoded string */
void cpdf_setCreationDate(int, char*);

/* Set the modification date of a document from a UTF8 encoded string */
void cpdf_setModificationDate(int, char*);

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

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

/* Get a box given the document, page range, min x, max x, min y, max y in
 * points. Only succeeds 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*);
```

```

/* 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);

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

/* Mark a document as untrapped. */
void cpdf_markUntrapped(int);

/* Document Layouts. See ISO standard for details. */
enum cpdf_layout
{cpdf_singlePage,
 cpdf_oneColumn,
 cpdf_twoColumnLeft,
 cpdf_twoColumnRight,
 cpdf_twoPageLeft,
 cpdf_twoPageRight};

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

/* Document page modes. See ISO standard for details. */
enum cpdf_pageMode
{cpdf_useNone,
 cpdf_useOutlines,
 cpdf_useThumbs,
 cpdf_useOC,
 cpdf_useAttachments};

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

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

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

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

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

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

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

```

11. DOCUMENT INFORMATION AND METADATA

```
/* Set the XML metadata of a document, given a file name */
void cpdf_setMetadataFromFile(int, char*);

/* Set the XML metadata from a byte array. cpdf_setMetadataFromByteArray(pdf,
 * data, length) uses length characters from data. */
void cpdf_setMetadataFromByteArray(int, void*, int);

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

/* Remove the XML metadata from a document */
void cpdf_removeMetadata(int);
```

12 File Attachments

```
/* Attach a file, given its file name, and the pdf. It is attached at document
* level. */
void cpdf_attachFile(char*, int);

/* Attach a file, given its file name, pdf, and the page number to which it
* should be attached. */
void cpdf_attachFileToPage(char*, int, int);

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

/* List information about attachments. Call cpdf_startGetAttachments first,
* then cpdf_startGetAttachments 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);
int cpdf_numberGetAttachments(void);
char* cpdf_getAttachmentName(int);
void cpdf_endGetAttachments(void);
```

13 Miscellaneous

```
/* Make a draft document. The first argument is the document, second the range,
 * third is a boolean -- true to replace images with boxes, false to replace
 * them with nothing. */
void cpdf_draft(int, int, int);

/* Blacken all text in the given document and range */
void cpdf_blackText(int, int);

/* Blacken all lines in the given document and range */
void cpdf_blackLines(int, int);

/* Blacken all fills in the given document and range */
void cpdf_blackFills(int, int);

/* Thicken lines. cpdf_thinLines(pdf, range, min_thickness) thickens every line
 * less than min_thickness to min_thickness */
void cpdf_thinLines(int, int, double);

/* Copy the /ID from the first document to the second. */
void cpdf_copyId(int, int);
```

14 Page Labels

```
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 set of page labels.

cpdf_addPageLabels(pdf, style, prefix, offset, range)

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, char*, int, int);
```

15 Special functionality 1 – 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};  
  
/* Return the status of a PDF */  
enum cpdf_pdfStatus cpdf_lookupPdfStatus(int);  
  
/* Does a PDF have a given permission, assuming it is or was encrypted? */  
int cpdf_hasPermissionStatus(int, enum cpdf_permission);  
  
/* What is (or was) the encryption method? */  
enum cpdf_encryptionMethod cpdf_lookupPdfEncryption(int);  
  
/* Find the user password which was used to decrypt a PDF, if it has status  
 * cpdf_wasDecryptedWithUser */  
char* cpdf_lookupPdfUserPassword(int);
```

16 Special functionality 2 – Undo

```
/* Cpdf can hold multiple versions of each PDF, sharing data between them to
 * save memory. */

/* Mark a document for update. This copies the document so the change can be
 * undone later */
void cpdf_aboutToUpdate(int);

/* Same, but when a deep copy (no sharing of data) is required. */
void cpdf_aboutToUpdateDeep(int);

/* Abort such an update due to an error part-way through the update */
void cpdf_abortUpdate(int);

/* Undo a document. Returns true if managed to undo, false if nothing to undo
 * to. */
int cpdf_undo(int);

/* Redo a document. Returns true if managed to redo, false if nothing to redo
 * to. */
int cpdf_redo(int);
```

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 "cpdfplibwrapper.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;
}
```

