
ClibPDF Library Reference Manual

V2.02 Addendum

[Manual version 2.02-Addendum; 1999-12-09]

Copyright ©1999 FastIO™ Systems, Inc. All Rights Reserved.

ClibPDF™ is a library of C functions for generating PDF files directly. This is a supplement/addendum to the **ClibPDF™** manual version 2.01-r2 dated October 16, 1999, until the sections below are incorporated into the main manual.

New for TextBox

TextBox functions `cpdf_textBox()`, `cpdf_textBoxY()` and their "raw" versions now honor the **form-feed** (**FF** character or `'\f'`) in the text, provided that **NLmode** member in the (`CPDFtboxAttr *`) struct is non-zero. It will stop filling the text box when this character is encountered, and a pointer to any remaining text, if any, is returned by the functions. Therefore, the TEXT2PDF example will now perform a page break at each `'\f'`.

Unicode Support in Version 2.02-r1

Starting with version 2.01-r1 of ClibPDF, the use of Unicode text is supported for bookmarks (outlines), annotations, and to a limited extent in page content text. While Asian (Chinese, Japanese, and Korean) language text may be specified for page content without using Unicode, you *must* use Unicode to add annotations and bookmarks in these languages. Unicode strings require special handling because they typically contain many zero-valued bytes. This means that Unicode data cannot be passed as C strings in which a zero-valued byte

indicates the end of a string. Because ClibPDF API has been designed to accept text data as C strings, a workaround is needed to overcome this problem. We use a hexadecimal representation of Unicode strings for this purpose. By defining modes for the text API: *standard ASCII* or *Hexadecimal modes*, we are able to handle both standard strings and hexadecimal Unicode strings using a single set of text API functions.

Here is an example of adding two bookmarks, one in standard ASCII mode, and two additional bookmarks in Unicode. The third outline entry would be a typical usage where the Unicode data are passed to ClibPDF in binary form. ClibPDF offers a conversion function for converting binary Unicode data to and from a HEX representation.

Unicode Outline Examples

(Also See examples/unicode/unicode.c for another example.)

```
CPDFOutlineEntry *currOL;          /* pointer to current outline entry */
char hexbuf[2048];                 /* buffer for converted HEX string */

currOL = cpdf_addOutlineEntry(pdf, NULL, OL_SAME, OL_OPEN, pagenum,
    "Chapter 1: Introduction", DEST_Y, 792.0, p2, p3, p4);

cpdf_hexStringMode(pdf, YES);       /* Set HEX string mode */
currOL = cpdf_addOutlineEntry(pdf, currOL, OL_SAME, OL_OPEN, pagenum,
    "FEFF3057304A308AFF12FF1A30DA30FC30B8306E4E0B65B93078", DEST_Y, 200.0, p2, p3, p4);

/* Assume "*unicode" points to binary Unicode data of length "bytecount" */
currOL = cpdf_addOutlineEntry(pdf, currOL, OL_SAME, OL_OPEN, pagenum,
    cpdf_convertBinaryToHex(unicode, hexbuf, bytecount, 1), DEST_Y, 200.0, p2, p3, p4);

cpdf_hexStringMode(pdf, NO);        /* Reset HEX string mode before you forget */
```

Limitations of Unicode Use for Page Content Text

In the current version, you cannot yet use Unicode text with **cpdf_textAligned()** or any **TextBox** functions. This is because the string width function, **cpdf_stringWidth()** used internally to align text, does not work yet with Unicode. This should not be such a hardship for CJK text, because there are other non-Unicode encodings for which **cpdf_textAligned()** does work. **TextBox** functions cannot accept CJK text currently regardless of encodings. Please limit the use of Unicode for page content text to the following functions or their "raw" versions:

cpdf_text(), **cpdf_textShow()**, and **cpdf_textCRLFshow()**.

Also note, that the *byte-order detection bytes* "FEFF" appears unnecessary for Unicode strings for page content.

Line Feed in Annotations

Adobe Acrobat apparently uses the CR character (0x0D) to specify new lines in annotation text with both Unicode and standard ASCII. This is the default end-of-line (EOL) character for MacOS. It is likely that you will have to perform conversion of EOL characters for annotation text on Unix and Windows because they use different conventions for indicating EOL. Unix uses LF (0x0A) and Windows uses two characters CRLF (0x0D 0x0A). It is even possible that this conversion is necessary on MacOS as well because standard I/O functions such as `fread()` performs the EOL conversion to LF as required by the C standard.

Unicode Support Functions

void **cpdf_hexStringMode**(CPDFdoc *pdf, int flag);

This function sets the current mode for text API to be either HEX or standard ASCII. This call should be used with argument YES before HEX-coded Unicode strings are used in annotation, outline and basic text functions such as `cpdf_text()`. Before you can use ASCII strings again with the text API functions, the HEX mode must be reset.

`cpdf_hexStringMode(pdf, YES);`
places text API functions to accept HEX-coded strings.
`cpdf_hexStringMode(pdf, NO);`
must be used to restore standard ASCII-mode for strings.

char ***cpdf_convertBinaryToHex**(const unsigned char *datain, char *hexout, long length, int addFEFF);

This function converts binary data into a hexadecimal representation. A primary use of this function is to convert Unicode string to a hexadecimal string for passing to ClibPDF text API functions.

datain -- pointer to binary data of size "length"

hexout -- Converted HEX string is placed into this output buffer.

Buffer for "hexout" must be allocated by the caller with a storage for a string of size at least $2*length+5$ (considering possible addition of "FEFF" and string terminator).

length -- byte length of input binary data pointed to by datain.

addFEFF -- If non-zero, the function will check if "datain" begins with "FEFF" in binary and if not, prepend it to the output HEX representation. If it is zero, nothing is prepended.

unsigned char ***cpdf_convertHexToBinary**(const char *hexin, unsigned char *binout, long *length);

This function converts a hexadecimal string to a binary form. Non-HEX characters are simply skipped, e.g., CR, LF, space, comma, etc. may be present in input data and need not be removed before being passed to this function.

hexin -- HEX string input.

binout -- Converted binary data is placed into this output buffer.

The buffer for binout must be allocated by the caller, and must have the size of $\text{sizeof(hexin)}/2 + 1$ at a minimum.

length -- This variable will contain the byte length of converted binary data in "binout."

Additions and Changes that affect Premium ClibPDF Only

(NOTE: Similar capability is available as *cpdf_placeInLineImage()* in the base ClibPDF)

```
int cpdf_placeImageData(CPDFdoc *pdf, const char *uniqueID, const void *imagedata,
    long length, int nx, int ny, int ncomp_per_pixel, int bits_per_sample,
    float x, float y, float angle, float width, float height, int flags,
    CPDFimgAttr *imattr);
int cpdf_rawPlaceImageData(CPDFdoc *pdf, const char *uniqueID, const void *imagedata,
    long length, int nx, int ny, int ncomp_per_pixel, int bits_per_sample,
    float x, float y, float angle, float width, float height, int flags,
    CPDFimgAttr *imattr);
```

These two functions place bitmap image data that reside in memory (computed or preloaded from a file and decompressed) into a PDF via image XObject. They work in a similar way to **cpdf_placeInLineImage()**, but the difference is that the image data can be large and shared across multiple uses within a PDF file. In-line images are not shared across multiple instances.

const char *uniqueID -- Give a unique string identifying image data uniquely within a PDF file. This is used for keeping only one copy of image data for a PDF file, i.e., if the ID strings match for two calls to this function, it is assumed that the you are using the same image.

const void *imagedata -- This is your image data in uncompressed form. You must pass the image data as expected by PDF. If PDF requires byte alignment at the end of a scan line (I don't know), you must take care of it. Also, any byte or bit-order issues must be taken care of by you, the programmer. The function doesn't do anything in these areas. It simply compresses the data (by Flate if compression is ON), and inserts it into PDF.

long length ----- Number of bytes in image data.

int nx, ny -----Number of pixels in X and Y dimensions, respectively.

int ncomp_per_pixel -- Number of color components (e.g, RGB image should have 3 for this. For CMYK, it should be 4).

int bits_per_sample -- Number of bits per component (sample). For 8-bit gray image, this should be 8. For 24-bit RGB image (8 bit for each of RGB), this should be 8.

float x, y ----- Position of the lower-left corner of the image.

float width, height -- Size of the image in points (1/72 inches). NOTE: this is true for BOTH functions above.

int flags --- Bit-0: IM_GSAVE - (LSB) is "gsave" flag.
Usually this is needed. If you don't know, pass IM_GSAVE as flags.

Bit-1: IM_IMAGEMASK - if 1 and B/W images (1-bit per pixel), the image data is used as /ImageMask.

This should not be used with grayscale or color images.

Bit-2: IM_INVERT - if 1, gray or B/W sense is inverted (negative image).

Bit-3 and up: Reserved

Use above defines by OR'ing them, as in IM_GSAVE | IM_INVERT.

CPDFimgAttr *imattr ----- Always pass NULL for now.

Changes to Existing API Functions

int cpdf_importImage(CPDFdoc *pdf, const char *imagefile, int type, float x, float y, float angle, float *width, float *height, float *xscale, float *yscale, **int flags**);

int cpdf_rawImportImage(CPDFdoc *pdf, const char *imagefile, int type, float x, float y, float angle, float *width, float *height, float *xscale, float *yscale, **int flags**);

NOTE: the last argument of cpdf_importImage() and its "raw" version has been renamed to "**flags**" and its bit values are interpreted in the same way as described for cpdf_placeImageData(). All bits should work as described above, except that all flags other than IM_GSAVE will be ignored for PDFIMG (CPDF_IMG type).

[end of doc/1999-12-09;]