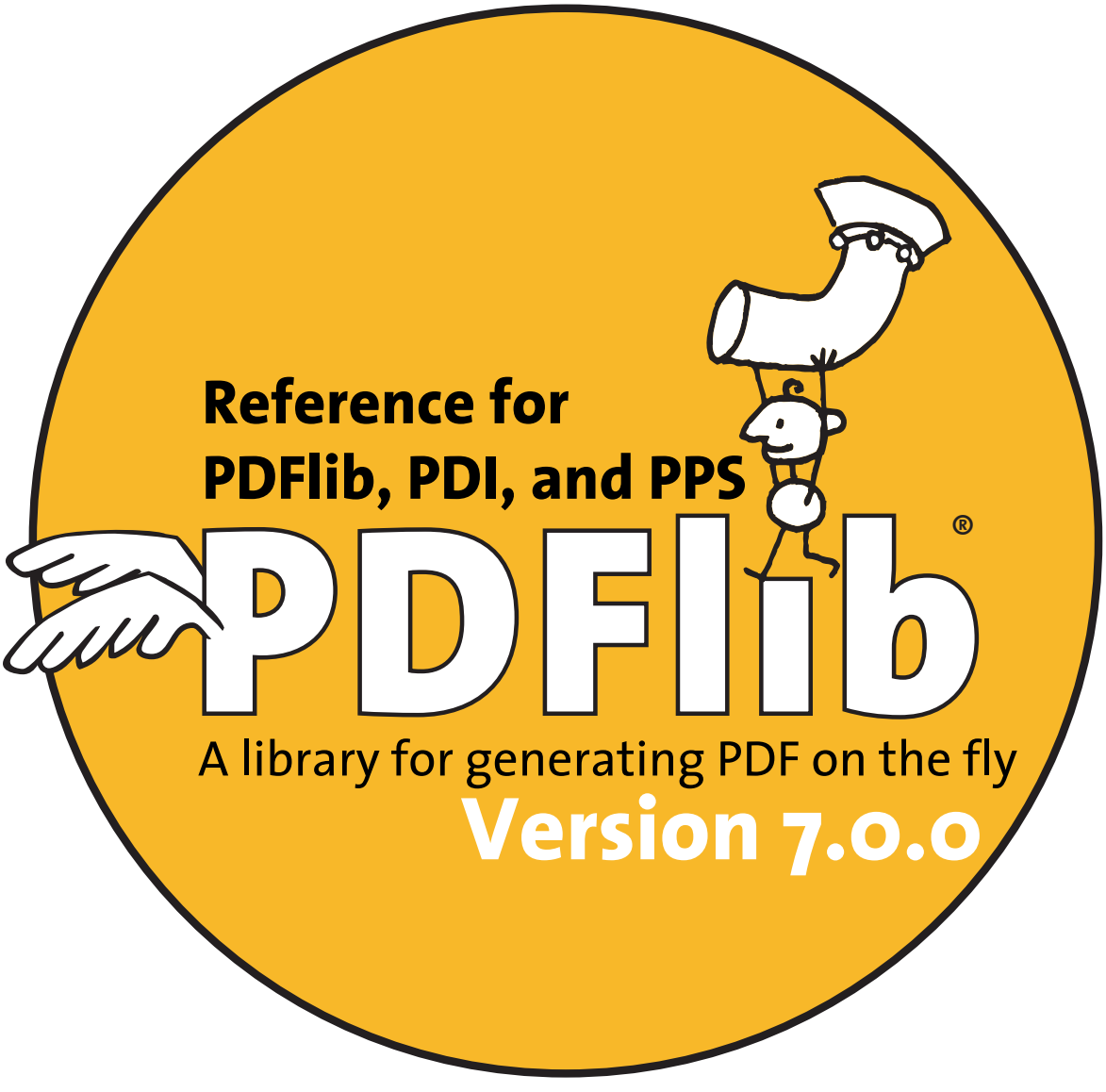
PDFlib GmbH München, Germany

www.pdflib.com



General Edition for
Cobol, C, C++, Java, Perl,
PHP, Python, RPG, Ruby, and Tcl

PDFlib GmbH
Tal 40, 80331 München, Germany
www.pdflib.com
phone +49 • 89 • 29 16 46 87
fax +49 • 89 • 29 16 46 86

If you have questions check the PDFlib mailing list and archive at tech.groups.yahoo.com/group/pdflib

Licensing contact: sales@pdflib.com
Support for commercial PDFlib licensees: support@pdflib.com (please include your license number)

Author: Thomas Merz
Design and illustrations: Alessio Leonardi
Quality control (manual): Katja Schnelle Romaus, Kurt Stützer
Quality control (software): a cast of thousands

Contents

1 PDFlib Programming Concepts

1.1 Data Types

This manual documents the function/method prototypes for various language bindings. The main difference between language bindings is that in object-oriented language bindings the PDFlib methods do not have the *PDF_* prefix in the name, while in other language bindings the *PDF_* prefix is part of all function names. Also, the PDF context parameter must be supplied as the first argument to all functions in non-object oriented language bindings. In contrast, the object-oriented language bindings hide the PDF context in an object created by the language wrapper.

Table 1.1 details the use of the PDF document type and the string data type in all language bindings. See the *PDFlib Tutorial* for more details on text and string handling. The data types *integer*, *long*, and *double* are not mentioned since there is an obvious mapping of these types in all bindings.

Table 1.1 Data types in the language bindings

language binding	p parameter?	PDF_ prefix?	string data type	binary data type
C (also used in this API reference)	yes	yes	const char * [1]	const char *
C++	no	no	string[2]	char *
Cobol	yes	no[3]	STRING	STRING
Java	no	no	String	byte[ ]
Perl	yes	yes	string	string
PHP	yes	yes	string	string
PHP 5 (object-oriented)	no	no	string	string
Python	yes	yes	string	string
RPG	yes	yes	string, but must add x'00'	data
Ruby	no	no	string	string
Tcl	yes	yes	string	byte array

1. C language NULL string values and empty strings are considered equivalent.
2. NULL string values must not be used in the C++ binding.
3. Cobol programs must use abbreviated names for the PDFlib functions.

1.2 Option Lists

Option lists are a powerful yet easy method for controlling API function calls. Instead of requiring a multitude of function parameters, many API methods support option lists, or *optlists* for short. These are strings which may contain an arbitrary number of options. Option lists support various data types and composite data like arrays. In most language bindings optlists can easily be constructed by concatenating the required keywords and values. C programmers may want to use the *sprintf( )* function to construct optlists. Table 1.2 lists various examples.

Table 1.2 Examples for options

option type	***example***
Float	`opacityfill=0.75`
Percentage	`leading=150%`
Boolean	`embedding` *(equivalent to* `embedding=true`)
Boolean	`nokerning` *(equivalent to* `kerning=false`)
String	`password { secret string }` *(the string value contains three blanks)*
String	`password {weird\}string}` *(the string value contains a right brace)*
Unichar	`replacementchar=space` *(equivalent to* `replacementchar=0x20`)
Keyword	`blendmode=overlay`
Rectangle	`cropbox={ 0 0 500 600 }`
List containing three numbers	`dasharray={ 11 22 33 }`
List containing two keywords	`position { center bottom }`
List containing a list	`polylinelist={ {10 20 30 40 } }`
Color	`backgroundcolor={ cmyk 0 1 0 0 }`
List containing a single action	`action={ activate { 0 1 2 } }`
List containing three actions	`action={ keystroke=0 format=1 validate=2 }`
Option list	`metadata={ filename=info.xmp }`
List containing one option list	`fill={ { area=table fillcolor={rgb 1 0 0} } }`
List containing two option lists	`fill = { { area=rowodd  fillcolor={rgb 0 1 0} }` `{ area=roweven fillcolor={rgb 1 0 0} } }`

Option list syntax. An optlist is a string containing one or more key/value pairs. Keys and values, as well as multiple key/value pairs can be separated by arbitrary whitespace characters (space, tab, carriage return, newline) or an equal sign '=':

```
key value
key=value
```

If the value is a string containing whitespace characters or equal signs you must surround the string with braces:

```
key={ multiple words }
key={ value=containing=equal=signs }
```

Since option lists will be evaluated from left to right an option can be supplied multiply within the same list. In this case the last occurrence will overwrite earlier ones. In the following example the first option assignment will be overridden by the second, and key will have the value 2 after processing the option list:

```
key=1 key=2
```

Option lists support the following data types which are discussed in more detail below:

- Simple values: boolean, string, content/hypertext/name string, unichar, keyword, float, integer, handle
- Composite values: lists, rectangles, action lists, color

For options of type list the list must be surrounded by braces, and the elements of the list must be separated by whitespace:

```
key={ value1 value2 value3 }
```

If an option list contains multiple key/value pairs make sure to insert whitespace before opening braces and after closing braces (instead of whitespace you can also use an equal character between the option name and the opening brace):

```
key1={ ...values... } key2={ ...values... }
```

Some options of type list accept the type *option list* or *list of option lists*. Options of type *option list* contain one or more subordinate options. Options of type *list of option lists* contain one or more nested option lists. When dealing with nested option lists it is important to specify the proper number of enclosing braces; see Table 1.2 for examples.

Simple Values. Simple values may use any of the following data types:

- Boolean: *true* or *false*; if the value of a boolean option is omitted, the value *true* is assumed. As a shorthand notation *noname* can be used instead of *name=false*.
- String: these are plain ASCII strings which are generally used for non-localizable keywords. Strings containing whitespace or '=' characters must be bracketed with *{* and *}*. An empty string can be constructed with *{ }*. The characters *{* and *}* must be preceded by an additional \ character if they are supposed to be part of the string.
- Content strings, hypertext strings and name strings: these can hold Unicode content in various formats; for details on these string types and encoding choices for string options see the *PDFlib Tutorial*.
- Unichar: these are single Unicode values where several syntax variants are supported: decimal values (e.g. 173), hexadecimal values prefixed with *x, X, ox, oX*, or U+ (xAD, 0xAD, U+00AD), numerical references, character references, and glyph name references but without the '&' and ';' decoration *(shy,* #xAD, #173). Alternatively, literal characters can be supplied. Unichars must be in the range 0-65535 (0-0xFFFF).
- Keyword: one of a predefined list of fixed keywords
- Float and integer: decimal floating point or integer numbers; point and comma can be used as decimal separators for floating point values. Integer values can start with *x, X, ox,* or *oX* to specify hexadecimal values. Some options (this is stated in the respective documentation) support percentages by adding a *%* character directly after the value.
- Handle: various types of object handles, e.g. font, image, or action handles. Technically these are integer values.

Depending on the type and interpretation of an option additional restrictions may apply. For example, integer or float options may be restricted to a certain range of values; handles must be valid for the corresponding type of object, etc. Conditions for options are documented in their respective function descriptions.

List Values. List values consist of multiple values, which may be simple values or list values in turn. Lists are bracketed with *{* and *}*.

Rectangles. A rectangle is a list of four float values specifying the coordinates of the lower left and upper right corners of a rectangle. The coordinate system for interpreting the rectangle coordinates (standard or user coordinate system) varies depending on the option, and is documented separately.

Action Lists. An action list specifies one or more actions. Each entry in the list consists of an event keyword (trigger) and a list of action handles which must have been created with *PDF_create_action( )*. Actions will be performed in the listed order. The set of allowed events (e.g. *docopen)* and the type of actions (e.g. JavaScript) are documented separately for the respective options.

Color. A color option is a list consisting of a color space keyword and a list with a variable number of float values depending on the particular color space. Color space keywords are the same as for *PDF_setcolor( )* (see Section 6.1, »Setting Color and Color Space«, page 101). Table 1.3 contains specific descriptions and examples. As detailed in the respective function descriptions, a particular option list may only supply a subset of the keywords presented above.

Table 1.3 Keywords for the color data type in option lists

keyword	***additional values***	***example***
gray	*single float value for the grayscale color space*	`{ gray 0.5 }`
rgb	*three float values for the RGB color space*	`{ rgb 1 0 0 }`
cmyk	*four float values for the CMYK color space*	`{ cmyk 0 1 0 0 }`
lab	*three float values for the Lab color space*	`{ lab 100 50 30 }`
spot	*spot color handle and a float specifying the tint value*	`{ spot 1 0.8 }`
spotname	*spot color name and a float specifying the tint value*	`{ spotname {PANTONE 281 U} 0.5 }`
spotname	*Similar to the simple form of* `spotname` *above, but a color value can be added to specify the alternate color for a custom spot color (i.e. a spot color name which is not known internally to PDFlib). If multiple options define the same custom spot color name all definitions must be consistent (i.e. define the same alternate color).*	`{ spotname {PDFlib Blue} 0.5 { lab 100 50 30 } }`
iccbasedgray	*single float value*	`{ iccbasedgray 0.5 }`
iccbasedrgb	*two float values*	`{ iccbasedrgb 1 0 0 }`
iccbasedcmyk	*three float values*	`{ iccbasedgray 0 1 0 0 }`
pattern	*pattern handle*	`{ pattern 1 }`
none	*specifies the absence of color*	`none`

1.3 Function Scopes

PDFlib applications must obey certain structural rules which are easy to understand. For example, you obviously begin a document before ending it. Since the PDFlib API is very closely modelled after the document/page paradigm, generating documents the »natural« way usually leads to well-formed PDFlib client programs.

PDFlib enforces correct ordering of function calls with a strict scoping system. The scope definitions can be found in Table 1.4. Figure 1.1 depicts the nesting of scopes. The function descriptions specify the allowed scope for all function. Calling a function outside of the allowed scopes will trigger a PDFlib exception. You can query the current scope with the *scope* parameter.

Table 1.4 Function scope definitions

scope name	*definition*
path	*started by one of PDF_moveto( ), PDF_circle( ), PDF_arc( ), PDF_arcn( ), or PDF_rect( ); terminated by any of the functions in Section 5.6, »Path Painting and Clipping«, page 95*
page	*between PDF_begin_page( ) and PDF_end_page(, but outside of* `path` *scope*
template	*between PDF_begin_template_ext( ) and PDF_end_template( ), but outside of* `path` *scope*
pattern	*between PDF_begin_pattern( ) and PDF_end_pattern() , but outside of* `path` *scope*
font	*between PDF_begin_font( ) and PDF_end_font( ), but outside of* `glyph` *scope*
glyph	*between PDF_begin_glyph( ) and PDF_end_glyph( ), but outside of* `path` *scope*
document	*between PDF_begin_document( ) and PDF_end_document(,) but outside of* `page`, `template`, `pattern`, *and* `font` *scope*
object	*in object-oriented language bindings: the lifetime of the pdflib object, but outside of* `document` *scope; in other bindings between PDF_new( ) and PDF_delete( ), but outside of* `document` *scope*
null	*outside of* `object` *scope*
any	*when a function description mentions »any« scope it actually means any except null, since a PDFlib object doesn't even exist in* `null` *scope.*

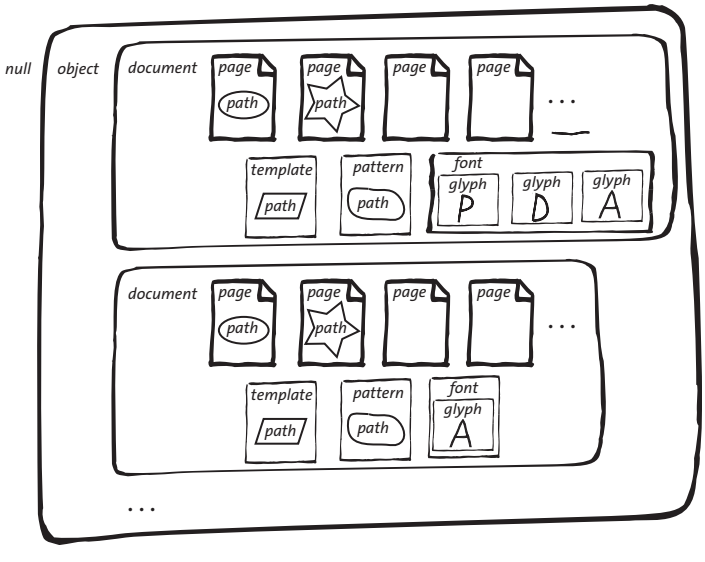


Fig. 1.1 Nesting of scopes

2 General Functions

2.1 Parameter Handling

PDFlib's operation can be controlled by a variety of global parameters. There are string parameters and numerical values for controlling PDFlib's operation and the appearance of the PDF output. Four functions are available for setting and retrieving numerical and string parameters. The descriptions of available keys can be found in the respective chapters; a summary of all supported parameters is available in Appendix B, »List of all Parameters«.

These parameters will retain their settings across the life span of the PDFlib object, or until they are explicitly changed by the client. However, some parameters will explicitly be reset at the beginning of each page (this is mentioned in the respective descriptions).

C++ Java ***double get_value(String key, double modifier)***

Perl PHP ***float PDF_get_value(resource p, string key, float modifier)***

C ***double PDF_get_value(PDF *p, const char *key, double modifier)***

Get the value of some PDFlib parameter with numerical type.

key The name of the parameter to be queried.

modifier An optional modifier to be applied to the parameter. Whether a modifier is required and what it relates to is explained in the various parameter tables. If the modifier is unused it must be 0. Many parameters require handles to be passed as modifier.

Returns The numerical value of the parameter.

Scope Depends on *key*.

C++ Java ***void set_value(String key, double value)***

Perl PHP ***PDF_set_value(resource p, string key, float value)***

C ***void PDF_set_value(PDF *p, const char *key, double value)***

Set the value of some PDFlib parameter with numerical type.

key The name of the parameter to be set.

value The new value of the parameter to be set.

Scope Depends on *key*.

C++ Java ***String get_parameter(String key, double modifier)***

Perl PHP ***string PDF_get_parameter(resource p, string key, float modifier)***

C ***const char * PDF_get_parameter(PDF *p, const char *key, double modifier)***

Get the contents of some PDFlib parameter with string type.

key The name of the parameter to be queried.

modifier An optional modifier to be applied to the parameter. Whether a modifier is required and what it relates to is explained in the various parameter tables. If the modifier is unused it must be 0.

Returns The string value of the parameter as a hypertext string. The returned string can be used until the end of the surrounding *document* scope. If no information is available an empty string will be returned.

Scope Depends on *key*.

Bindings C and C++: C and C++ clients must not free the returned string. PDFlib manages all string resources internally.

C++ Java ***void set_parameter(String key, String value)***

Perl PHP ***PDF_set_parameter(resource p, string key, string value)***

C ***void PDF_set_parameter(PDF *p, const char *key, const char *value)***

Set some PDFlib parameter with string type.

key The name of the parameter to be set.

value (Name string) The new value of the parameter to be set.

Scope Depends on *key*.

2.2 Setup

Table 2.1 and Table 2.2 list relevant parameter and value key names for PDFlib setup (see Section 2.1, »Parameter Handling«, page 11).

Table 2.1 Setup-related keys for PDF_get/set_parameter( )

key	explanation
any resource category name	*Entries in any of the resource categories. PDF_get_parameter( ): Modifier contains the index of the entry (starting with 1). If there are no more entries an empty string will be returned. See PDFlib Tutorial for a list of category names. Scope: any*
asciifile	*(Only supported on iSeries and zSeries). Expect text files (PFA, AFM, UPR, encodings) in ASCII encoding. Default: true on iSeries; false on zSeries. Scope: any*
license[1]	*Set the license key for PDFlib, PDFlib+PDI, or PPS.The key can be set (even multiply to accumulate keys) before the first call to PDF_begin_document( ). Scope: object*
licensefile	*Set the name of a file containing the license key.The license file can only be set once before the first call to PDF_begin_document( ). Scope: object*
nodemo-stamp	*If true, an exception will be thrown when no valid license key was found; if false, a demo stamp will be created on all pages. This option must be set before the first call to PDF_begin_document( ). Default:* `false`*. Scope: object*
resourcefile	*Relative or absolute file name of the PDFlib UPR resource file. The resource file will be loaded immediately. Existing resources will be kept; their values will be overridden by new ones if they are set again. Scope: any*
scope[1]	*Name of the current scope (see Table 1.4). Scope: any*
SearchPath	*(Not supported on MVS) Relative or absolute path name of a directory containing files to be read. The SearchPath can be set multiply; the entries will be accumulated and used in least-recently-set order. An empty string deletes all entries from the SearchPath list.* *PDF_get_parameter( ): Modifier contains the index of the entry (starting with 1). If there are no more entries an empty string will be returned. The returned string will be encoding in UTF-8. Scope: any*
string[1]	*Return a string identified by the string index supplied in the modifier. The returned string is valid until the next call to any API function. Scope: any*
version[1]	*Full PDFlib version string in the format <major>.<minor>.<revision>, possibly suffixed with additional qualifiers such as beta, rc, etc. Scope: any, null*[2]

1. Only for PDF_get_parameter( )
*2. May be called with a PDF * argument of NULL or 0*

Table 2.2 Keys for PDF_get/set_value( ) (see Section 2.1, »Parameter Handling«, page 11)

key	explanation
compress	*Compression level from 0=no compression, 1=best speed, etc. to 9=best compression. This parameter does not affect image data handled in passthrough mode. Default: 6. Scope: page, document*
major minor revision[1]	*Major, minor, or revision number of PDFlib, respectively. Scope: any, null*[2]

1. Only for PDF_get_value( )
*2. May be called with a PDF * argument of NULL or 0*

Perl PHP ***resource PDF_new( )***

C ***PDF *PDF_new(void)***

Create a new PDFlib object.

Details This function creates a new PDFlib object, using PDFlib's internal default error handling and memory allocation routines.

Returns A handle to a PDFlib object which is to be used in subsequent PDFlib calls. If this function doesn't succeed due to unavailable memory it will return NULL (in C) or throw an exception.

Scope *null;* this function starts object scope, and must always be paired with a matching *PDF_delete( )* call.

Bindings The data type used for the opaque PDFlib object handle varies among language bindings. This doesn't really affect PDFlib clients, since all they have to do is pass the PDF handle as the first argument to all functions.

C: In order to load the PDFlib DLL dynamically at runtime use *PDF_new_dl( )*. *PDF_new_dl( )* will return a pointer to a *PDFlib_api* structure filled with pointers to all PDFlib API functions. If the DLL cannot be loaded, or a mismatch of major or minor version number is detected, NULL will be returned.
C++, Java, PHP 5: this function is not available since it is hidden in the PDFlib constructor.

C ***PDF *PDF_new2(void (*errorhandler)(PDF *p, int errortype, const char *msg),***
void* (*allocproc)(PDF *p, size_t size, const char *caller),
void* (*reallocproc)(PDF *p, void *mem, size_t size, const char *caller),
void (*freeproc)(PDF *p, void *mem),
void *opaque)

Create a new PDFlib object with client-supplied error handling and memory allocation routines.

errorhandler Pointer to a user-supplied error-handling function. The error handler will be ignored in *PDF_TRY/PDF_CATCH* blocks.

allocproc Pointer to a user-supplied memory allocation function.

reallocproc Pointer to a user-supplied memory reallocation function.

freeproc Pointer to a user-supplied free function.

opaque Pointer to some user data which may be retrieved later with *PDF_get_opaque( )*.

Returns A handle to a PDFlib object which is to be used in subsequent PDFlib calls. If this function doesn't succeed due to unavailable memory it will return NULL (in C) or throw an exception.

Details This function creates a new PDFlib object with client-supplied error handling and memory allocation routines. Unlike *PDF_new( )*, the caller may optionally supply own procedures for error handling and memory allocation. The function pointers for the error

handler, the memory procedures, or both may be NULL. PDFlib will use default routines in these cases. Either all three memory routines must be provided, or none.

Scope *null;* this function starts *object* scope, and must always be paired with a matching *PDF_delete( )* call. No other PDFlib function with the same PDFlib object must be called after calling this function.

Bindings C++: this function is indirectly available via the PDF constructor. Not all function arguments must be given since default values of NULL are supplied. All supplied functions must be »C« style functions, not C++ methods.

Perl PHP ***PDF_delete(resource p)***

C ***void PDF_delete(PDF *p)***

Delete a PDFlib object and free all internal resources.

Details This function deletes a PDF object and frees all document-related PDFlib-internal resources. Although not necessarily required for single-document generation, deleting the PDF object is heavily recommended for all server applications when they are done producing PDF. This function must only be called once for a given PDF object. *PDF_delete( )* should also be called for cleanup when an exception occurred. *PDF_delete( )* itself is guaranteed to not throw any exception. If more than one PDF document will be generated it is not necessary to call *PDF_delete( )* after each document, but only when the complete sequence of PDF documents is done.

Scope *any*; this function starts *null* scope, i.e. no more API function calls are allowed.

Bindings C: If the PDFlib DLL has been loaded dynamically at runtime with *PDF_new_dl( ),* use *PDF_delete_dl( )* to delete the PDFlib object.

C++: this function is indirectly available via the PDF destructor.
Java: this function is automatically called by the wrapper code. However, it can explicitly be called from client code in order to overcome shortcomings in Java's finalizer system.
PHP: this function will automatically be called for the object-oriented PHP 5 interface when the PDFlib object goes out of scope.

2.3 Document Functions

C++ Java ***int begin_document(String filename, String optlist)***

Perl PHP ***int PDF_begin_document(resource p, string filename, string optlist)***

C ***int PDF_begin_document(PDF *p, const char *filename, int len, const char *optlist)***

C++ ***void begin_document_callback(size_t (*writeproc) (PDF *p, void *data, size_t size), string optlist)***

C ***void PDF_begin_document_callback(PDF *p,***
size_t (*writeproc) (PDF *p, void *data, size_t size), const char *optlist)

Create a new PDF document subject to various options.

filename (Name string, but Unicode file names are only supported on Windows) Absolute or relative name of the PDF output file to be generated. If *filename* is empty, the PDF document will be generated in memory instead of on file, and the generated PDF data must be fetched by the client with the *PDF_get_buffer( )* function. The special file name »–« can be used for generating PDF on the *stdout* channel. On Windows it is OK to use UNC paths or mapped network drives.

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

writeproc (Only for C and C++) C callback function which will be called by PDFlib in order to submit (portions of) the generated PDF data.

optlist An option list specifying document options according to Table 2.3. Options specified in *PDF_end_document( )* have precedence over identical options specified in *PDF_begin_document( )*. The following options can be used:
action, attachments, autoxmp, compatibility, destination, destname, errorpolicy, flush, groups, hypertextencoding, inmemory, labels, lang, linearize, masterpassword, metadata, moddate, openmode, optimize, pagelayout, pdfa, pdfx, permissions, recordsize, search, tagged, tempdirname, tempfilenames, uri, userpassword, viewerpreferences.

Returns -1 (in PHP: 0) on error, and 1 otherwise. If *filename* is empty this function will always succeed, and never return the -1 (in PHP: 0) error value.

Details This function creates a new PDF file using the supplied *filename*. PDFlib will attempt to open a file with the given name, and close the file when the PDF document is finished.

PDF_begin_document_callback( ) opens a new PDF document in memory, without writing to a disk file. The callback function supplied as *writeproc* must return the number of bytes written. If the return value doesn't match the *size* argument supplied by PDFlib, an exception will be thrown. The frequency of *writeproc* calls is configurable with the *flush* option.

Scope *object;* this function starts *document* scope if the file could successfully be opened, and must always be paired with a matching *PDF_end_document( )* call.

Bindings C, C++, Java, JScript: take care of properly escaping the backslash path separator. For example, the following denotes a file on a network drive: *\\\\malik\\rp\\foo.pdf*.
PDF_begin_document_callback( ) is only available in C and C++. The supplied *writeproc* must be a C-style function, not a C++ method.

C++ Java ***void end_document(String optlist)***

Perl PHP ***PDF_end_document(resource p, string optlist)***

C ***void PDF_end_document(PDF *p, const char *optlist)***

Close the generated PDF document and apply various options.

optlist An option list specifying document options according to Table 2.3. Options specified in *PDF_end_document( )* have precedence over identical options specified in *PDF_begin_document( )*. The following options can be used:
action, attachments, autoxmp, destination, destname, hypertextencoding, labels, metadata, moddate, openmode, pagelayout, search, uri, viewerpreferences.

Details This function finishes the generated PDF document, frees all document-related resources, and closes the output file if the PDF document has been opened with *PDF_begin_document( )*. This function must be called when the client is done generating pages, regardless of the method used to open the PDF document.

When the document was generated in memory (as opposed to on file), the document buffer will still be kept after this function is called (so that it can be fetched with *PDF_get_buffer( )*), and will be freed in the next call to *PDF_begin_document( )*, or when the PDFlib object goes out of scope in *PDF_delete( )*.

Scope *document;* this function terminates *document* scope, and must always be paired with a matching call to one of *PDF_begin_document( ) or PDF_begin_document_callback( ).*

Table 2.3 Document options for PDF_begin_document( ) and PDF_end_document( )

option	description
action[1]	*(Action list; not allowed for PDF/A) List of document actions for one or more of the following events. Default: empty list.* ***open*** *Actions to be performed when the document is opened. Due to the execution order in Acrobat document-level JavaScript must not be used for open actions.* ***didprint/didsave/willclose/willprint/willsave*** *(PDF 1.4) JavaScript actions to be performed after printing/after saving/before closing/before printing/ before saving the document.*
attachments	*(List of option lists) Specifies document-level file attachments (as opposed to attachment annotations which are bound to a particular location on a page). It is ok to supply file attachments both in PDF_begin_document( ) and PDF_end_document( ). Supported options:* ***filename*** *(Name string; required) Name of the file. UTF-16 file names are supported.* ***description*** *(Hypertext string; PDF 1.6) Descriptive text associated with the file.*
autoxmp	*(Boolean; will be forced to* `true` *in PDF/A mode) If* `true`*, PDFlib will create XMP document metadata from document info fields (see Section 12.2, »XMP Metadata«, page 161). Default:* `false`
compatibility[2]	*(Keyword) Set the document's PDF version to one of the strings* `1.3`*,* `1.4`*,* `1.5`*,* `1.6`*, or* `1.7` *for Acrobat 4, 5, 6, 7, or 8. This option will be ignored if one of the* `pdfx` *or* `pdfa` *options is used. Default:* `1.6`
destination	*(Option list; will be ignored if an open action has been specified) An option list specifying the document open action according to Table 10.3.*
destname[1]	*(Hypertext string; will be ignored if the* `destination` *option has been specified) The name of a destination which has been defined with PDF_add_nameddest( ), and will be used as the document open action.*
errorpolicy[2]	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*

Table 2.3 Document options for PDF_begin_document( ) and PDF_end_document( )

option	description
flush[2]	*(Keyword; only for PDF_begin_document_callback( )) Set the flushing strategy. Default:* page. **none** *flush only once at the end of the document* **page** *flush at the end of each page* **content** *flush after all fonts, images, file attachments, and pages* **heavy** *always flush when the internal 64 KB document buffer is full*
groups[2]	*(List of strings) Define the names and ordering of the page groups used in the document. Page groups keep pages together (useful e.g. for attaching page labels); pages can be assigned to one of the page groups defined in the document, and referenced within the respective group. If page groups are defined for a document, all pages must be a assigned to a page group.*
hypertext-encoding	*(Keyword) Specifies the encoding for the* destname *option (see PDFlib Tutorial). An empty string is equivalent to unicode. Default: value of the global* hypertextencoding *parameter.*
inmemory[2]	*(Boolean; not for PDF_begin_document_callback( )) If* true *and the* linearize *or* optimize *option is* true *as well, PDFlib will not create any tempory files for linearization, but will process the file in memory. This can result in tremendous performance gains on some systems (especially MVS), but requires memory twice the size of the document. If false, a temporary file will be created for linearization and optimization. Default:* false
labels	*(List of option lists) A list containing one or more option lists according to Table 2.4 specifying symbolic page names. The page name will be displayed as a page label (instead of the page number) in Acrobat's status line. The combination of style/prefix/start values must be unique within a document. Default: none*
lang[2]	*(String; required if* tagged=true*) Set the natural language of the document as a two-character ISO 639 language code (examples: DE, EN, FR, JA), optionally followed by a hyphen and a two-character ISO 3166 country code (examples: EN-US, EN-GB, ES-MX). Case is not significant.* *The language specification can be overridden for individual items on all levels of the structure tree, but must be set initially for the document as a whole.*
linearize[2]	*(Boolean; not for PDF_begin_document_callback( )) If* true*, the output document will be linearized. On MVS systems this option cannot be combined with in-core generation (i.e. an empty filename). Default:* false
master-password[2]	*(String; not for PDF/A and PDF/X) The master password for the document. If it is empty no master password will be applied. Default: empty*
metadata	*(Option list; PDF 1.4) Supply document metadata (see Section 12.2, »XMP Metadata«, page 161)*
moddate	*(Boolean) If* true*, the* ModDate *(modification date) document info key will be created for compliance with some preflight tools. Default:* false
openmode	*(Keyword) Set the appearance when the document is opened. Default:* bookmarks *if the document contains any bookmarks, otherwise* none*:* **none** *Open with no additional panel visible.* **bookmarks** *Open with the bookmark panel visible.* **thumbnails** *Open with the thumbnail panel visible.* **fullscreen** *Open in fullscreen mode (does not work in the browser).* **layers** *(PDF 1.5) Open with the layer panel visible.* **attachments** *(PDF 1.6) Open with the attachments panel visible.*
optimize[2]	*(Boolean) If* true*, the output document will be optimized in a separate pass after generating it. Optimization reduces file size by eliminating redundant duplicate objects. On MVS systems this option cannot be combined with in-core generation (i.e. an empty filename). Default:* false

Table 2.3 Document options for PDF_begin_document( ) and PDF_end_document( )

option	description
pagelayout	*(Keyword) The page layout to be used when the document is opened. Default:* default.
	default *The default setting of the Acrobat viewer.*
	singlepage *Display one page at a time.*
	onecolumn *Displays the pages continously in one column.*
	twocolumnleft *Display the pages in two columns, odd pages on the left.*
	twocolumnright *Display the pages in two columns, odd pages on the right*
	twopageleft *(PDF 1.5) Display the pages two at a time, with odd-numbered pages on the left.*
	twopageright *(PDF 1.5) Display the pages two at a time, with odd-numbered pages on the right.*
pdfa[2]	*(Keyword) Set the PDF/A conformance level to one of* PDF/A-1a:2005, PDF/A-1b:2005, *or* none. *The value »PDF/A-1a:2005« will automatically enable Tagged PDF mode. Default:* none
pdfx[2]	*(Keyword) Set the PDF/X conformance level to one of* PDF/X-1a:2001, PDF/X-1a:2003, PDF/X-2:2003, PDF/X-3:2002, PDF/X-3:2003, *or none. Default:* none
permissions[2]	*(Keyword list; not for PDF/A and PDF/X) The access permission list for the output document. It contains any number of the following keywords (default: empty).*
	noprint *Acrobat will prevent printing the file.*
	nomodify *Acrobat will prevent users from adding form fields or making any other changes.*
	nocopy *Acrobat will prevent copying and extracting text or graphics, and will disable the accessibility interface*
	noannots *Acrobat will prevent adding or changing comments or form fields.*
	noforms *(PDF 1.4) Acrobat will prevent form field filling, even if noannots hasn't been specified.*
	noaccessible *(PDF 1.4) Acrobat will prevent extracting text or graphics for accessibility purposes (such as a screenreader program)*
	noassemble *PDF 1.4) Acrobat will prevent inserting, deleting, or rotating pages and creating bookmarks and thumbnails, even if nomodify hasn't been specified.*
	nohiresprint *(PDF 1.4) Acrobat will prevent high-resolution printing. If noprint hasn't been specified printing is restricted to the »print as image« feature which prints a low-resolution rendition of the page.*
	plainmetadata *(PDF 1.5) Keep XMP document metadata unencrypted even for encrypted documents*
recordsize[2]	*(Integer; MVS only) The record size of the output file (and any temporary file which may have to be created for the* linearize *and* optimize *options. Default: 0 (unblocked output)*
search	*(Option list) Instruct Acrobat to attach a search index when opening the document. The following suboptions are supported:*
	filename *(Name string; required) The name of a file containing a search index. The file name of the index may be relative to the document, but the user is responsible for supplying correct index file names.*
	indextype *(Name string) The type of the index; must be* PDX *for Acrobat. Default:* PDX
tagged[2]	*(Boolean; PDF 1.4) If* true, *generate Tagged PDF output. Proper structure information must be provided by the client in Tagged PDF mode (see Section 12.3, »Tagged PDF«, page 163). If the* pdfa *option has the value »PDF/A-1a:2005« this option will automatically be forced to true. Default:* false
tempdirname[2]	*(String; not for PDF_begin_document_callback( )) Name of a directory where temporary files required for the* linearize *and* optimize *options will be created. If empty, PDFlib will generate temporary files in the current directory. This option will be ignored if the* tempfilenames *option has been supplied. Default: empty*

Table 2.3 Document options for PDF_begin_document( ) and PDF_end_document( )

option	description
temp-filenames[2]	*(List of two strings; only on MVS and for PDF_begin_document( )) Full file names for two temporary files required for the* `linearize` *and* `optimize` *options. If empty, PDFlib will generate unique temporary file names. The user is responsible for deleting the temporary files after PDF_end_document( ). If this option is supplied the* `filename` *parameter must not be empty. Default: empty*
uri	*(String) Set the document's base URL. This is useful when a document with relative Web links to other documents is moved to a different location. Setting the base URL to the »old« location makes sure that relative links will still work. Default: none*
user-password[2]	*(String; not for PDF/A and PDF/X) The user password for the document. If it is empty no user password will be applied. Default: empty*
viewer-preferences	*(Option list) Option list specifying various viewer preferences according to Table 2.5. Default: empty*

1. Only for PDF_end_document( )

Table 2.4 Suboptions for the `labels` *option in PDF_begin/end_document( ) and* `label` *option in PDF_begin/end_page_ext( )*

option	description
group	*(String; only for PDF_begin_document( ); required if the document uses page groups, but not allowed otherwise) The label will be applied to all pages in the specified group and all pages in all subsequent groups until a new label is applied. The group name must have been defined with the groups option in PDF_begin_document( ).*
hypertext-encoding	*(Keyword) Specifies the encoding for the* `prefix` *option. An empty string is equivalent to* `unicode`*. Default: value of the global* `hypertextencoding` *parameter.*
pagenumber	*(Integer; only for PDF_end_document( ); required if the document does not use page groups, but not allowed otherwise) The label will be applied to the specified page and subsequent pages until a new label is applied.*
prefix	*(Hypertext string) The label prefix for all labels in the range. Default: none*
start	*(Integer >= 1) Numeric value for the first label in the range. Subsequent pages in the range will be numbered sequentially starting with this value. Default: 1*
style	*(Keyword) The numbering style to be used. Default:* `none`*.* **none** *no page number; labels will only consist of the prefix.* **D** *decimal arabic numerals (1, 2, 3, ...)* **R** *uppercase roman numerals (I, II, III, ...)* **r** *lowercase roman numerals (i, ii, iii, ...)* **A** *uppercase letters (A, B, C, ..., AA, BB, CC, ...)* **a** *lowercase letters (a, b, c, ..., aa, bb, cc, ...)*

Table 2.5 Suboptions for the viewerpreferences *option in PDF_begin_document( ) and PDF_end_document( )*

option	*description*
centerwindow	*(Boolean) Specifies whether to position the document's window in the center of the screen. Default:* false
direction	*(Keyword) The reading order of the document, which affects the scroll ordering in double-page view. (default* l2r*):* ***l2r*** *Left to right* ***r2l*** *Right to left (including vertical writing systems)*
displaydoctitle	*(Boolean) Specifies whether to display the Title document info field in Acrobat's title bar (*true*) or the file name (*false*). Default:* false
fitwindow	*(Boolean) Specifies whether to resize the document's window to the size of the first page. Default:* false
hidemenubar	*(Boolean) Specifies whether to hide Acrobat's menu bar. Default:* false
hidetoolbar	*(Boolean) Specifies whether to hide Acrobat's tool bars. Acrobat 5 ignores this setting when viewing PDFs in a browser. Default:* false
hidewindowui	*(Boolean) Specifies whether to hide Acrobat's window controls. Default:* false
nonfullscreen-pagemode	*(Keyword; only relevant if the* openmode *option is set to fullscreen) Specifies how to display the document on exiting full-screen mode. Default:* none. ***bookmarks*** *display page and bookmark pane* ***thumbnails*** *display page and thumbnail pane* ***layers*** *display page and layer pane* ***none*** *display page only*
printscaling	*(Keyword; PDF 1.6) Page scaling option to be selected when a print dialog is presented for the document. Default:* appdefault. ***none*** *No page scaling; this may be useful for printing page contents at their exact sizes.* ***appdefault*** *Use the current print scaling as specified in Acrobat.*
viewarea ***viewclip*** ***printarea*** ***printclip***	*(Keyword) The type of the page boundary box representing the area of a page to be displayed or clipped when viewing the document on screen or printing it. Acrobat ignores this setting, but it may be useful for other applications. Default:* crop. ***art*** *Use the ArtBox* ***bleed*** *Use the BleedBox* ***crop*** *Use the CropBox* ***media*** *Use the MediaBox* ***trim*** *Use the TrimBox* *PDF/X: values other than* media *or* bleed *are not allowed.*

C++ Java ***byte[] get_buffer( )***

Perl PHP ***string PDF_get_buffer(resource p)***

C ***const char * PDF_get_buffer(PDF *p, long *size)***

Get the contents of the PDF output buffer.

size (C and C++ language bindings only) C-style pointer to a memory location where the length of the returned data in bytes will be stored.

Returns A buffer full of binary PDF data for consumption by the client. It returns a language-specific data type for binary data according to Table 1.1. The returned buffer must be used by the client before calling any other PDFlib function. Remember to copy the data if you want to use it while calling other PDFlib functions (in particular, before calling *PDF_create_pvf( )* to create a PVF file containing the data).

Details Fetch the full or partial buffer containing the generated PDF data. If this function is called between page descriptions, it will return the PDF data generated so far. If generating PDF into memory, this function must at least be called after *PDF_end_document( )*, and will return the remainder of the PDF document. It can be called earlier to fetch partial document data. If there is only a single call to this function which happens after *PDF_end_document( )* the returned buffer is guaranteed to contain the complete PDF document in a contiguous buffer.

Since PDF output contains binary characters, client software must be prepared to accept non-printable characters including null values.

Scope *object, document* (in other words: after *PDF_end_page_ext( )* and before *PDF_begin_page_ext( )*, or after *PDF_end_document( )* and before *PDF_delete( )*. This function can only be used if an empty filename has been supplied to *PDF_begin_document( )*.

If the *linearize* option in *PDF_begin_document( )* has been set to *true*, the scope is restricted to *object*, i.e. this function can only be called after *PDF_end_document( )*.

Bindings C and C++: the *size* parameter is only used for C and C++ clients.

Other bindings: an object of appropriate length will be returned, and the *size* parameter must be omitted.

2.4 Page Functions

Table 2.6 and Table 2.7 list relevant parameter and value key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 2.6 Page-related keys for PDF_get/set_parameter( )

key	explanation
topdown	*If* `true`*, the origin of the coordinate system at the beginning of a page, pattern, or template will be assumed in the top left corner of the page, and y coordinates will increase downwards; otherwise the default coordinate system will be used. See PDFlib Tutorial for details. Scope: document. Default:* `false`

Table 2.7 Page-related keys for PDF_get/set_value( )

key	explanation
pagewidth **pageheight**	*Get the page size of the current page (dimensions of the MediaBox). Scope: page, path*

C++ Java ***void begin_page_ext(double width, double height, String optlist)***

Perl PHP ***PDF_begin_page_ext(resource p, float width, float height, string optlist)***

C ***void PDF_begin_page_ext(PDF *p, double width, double height, const char *optlist)***

Add a new page to the document and specify various options.

width, height The *width* and *height* parameters are the dimensions of the new page in points (or user units, if the *userunit* option has been specified). They can be overridden by the options with the same name (the dummy value 0 can be used for the parameters in this case). A list of commonly used page formats can be found in Table 2.8. See also Table 2.9 for more details (options *width* and *height)*.

Table 2.8 Common standard page size dimensions in points[1]

format	width	height	format	width	height	format	width	height
a0	2380	3368	a4	595	842	letter	612	792
a1	1684	2380	a5	421	595	legal	612	1008
a2	1190	1684	a6	297	421	ledger	1224	792
a3	842	1190	a5	501	709	11x17	792	1224

1. More information about ISO, Japanese, and U.S. standard formats can be found at the following URLs: home.inter.net/eds/paper/papersize.html, www.cl.cam.ac.uk/~mgk25/iso-paper.html

optlist An option list according to Table 2.9. These options have lower priority than identical options specified in *PDF_end_page_ext( )*. The following options can be used: *action, artbox, bleedbox, cropbox, defaultcmyk, defaultgray, defaultrgb, duration, group, height, label, metadata, mediabox, pagenumber, rotate, separationinfo, taborder, topdown, transition, trimbox, userunit, width*

Details This function will reset all text, graphics, and color state parameters for the new page to their defaults.

Scope *document;* this function starts *page* scope, and must always be paired with a matching *PDF_end_page_ext( )* call.

C++ Java ***void end_page_ext(String optlist)***

Perl PHP ***PDF_end_page_ext(resource p, string optlist)***

C ***void PDF_end_page_ext(PDF *p, const char *optlist)***

Finish a page and apply various options.

optlist An option list according to Table 2.9. Options specified in *PDF_end_page_ext( )* have priority over identical options specified in *PDF_begin_page_ext( )*. The following options can be used:
action, artbox, bleedbox, cropbox, defaultcmyk, defaultgray, defaultrgb, duration, group, height, label, mediabox, metadata, rotate, taborder, transition, trimbox, userunit, width

Scope *page;* this function terminates *page* scope, and must always be paired with a matching *PDF_begin_page_ext( )* call.

Table 2.9 Options for PDF_begin_page_ext( ) and PDF_end_page_ext( )

option	***description***
action	*(Action list) List of page actions for one or more of the following events (default: empty list):* ***open*** *Actions to be performed when the page is opened.* ***close*** *Actions to be performed when the page is closed.*
artbox ***bleedbox*** ***cropbox***	*(Rectangle) Change the page box parameters of the current page. The coordinates of the respective box are specified in the default coordinate system. By default, only the MediaBox will be created by using the* `width` *and* `height` *parameters.*
defaultgray ***defaultrgb*** ***defaultcmyk***	*(ICC handle) Set a default gray, RGB, or CMYK color space for the page according to the supplied profile handle.*
duration	*(Float) Set the page display duration in seconds for the current page if* `openmode=fullscreen` *(see Table 2.3). Default: 1*
group[1]	*(String; required if the document uses page groups, but not allowed otherwise) Name of the page group to which the page will belong. This name can be used to keep pages together in a page group and to address pages with PDF_resume_page( ). The group name must have been defined with the groups option in PDF_begin_document( ).*
height	*(Float or keyword; not allowed if the* `topdown` *option or parameter is* `true`*) The dimensions of the new page in points (or user units, if the* `userunit` *option has been specified). In order to produce landscape pages use* `width` > `height` *or the* `rotate` *option. PDFlib uses* `width` *and* `height` *to construct the page's MediaBox, but the MediaBox can also explicitly be set using the* `mediabox` *option. The* `width` *and* `height` *options will override the parameters with the same name.* *The following symbolic page size names can be used as keywords by appending* `.width` *or* `.height` *(e.g.* `a4.width`, `a4.height`*):* a0, a1, a2, ***a3***, a4, a5, a6, b5, `letter`, `legal`, `ledger`, `11x17`
label	*(Option list) An option list according to Table 2.4 specifying symbolic page names. The page name will be displayed as a page label (instead of the page number) in Acrobat's status line. The specified numbering scheme will be used for the current and subsequent pages until it is changed again. The combination of style/prefix/start values must be unique within a document.*

Table 2.9 Options for PDF_begin_page_ext( ) and PDF_end_page_ext( )

option	description
mediabox	*(Rectangle; not allowed if the* `topdown` *option or parameter is true) Change the page box parameters of the current page. The coordinates of the respective box are specified in the default coordinate system. By default, the MediaBox will be created by using the* `width` *and* `height` *parameters. The* `mediabox` *option will override the* `width` *and* `height` *options and parameters.*
metadata	*(Option list; PDF 1.4) Supply metadata for the page (see Section 12.2, »XMP Metadata«, page 161)*
pagenumber[1]	*(Integer) If this option is specified with a value n, the page will be inserted before the existing page n within the page group specified in the* `group` *option (or the document if the document doesn't use page groups). If this option is not specified the page will be inserted at the end of the group.*
rotate	*(Integer) The page rotation value. The rotation will affect page display, but does not modify the coordinate system. Possible values are 0, 90, 180, 270. Default: 0*
separation-info[1]	*(Option list) An option list containing color separation details for the current page. This will be ignored in Acrobat, but may be useful in third-party software for identifying and correctly previewing separated pages in a preseparated workflow:* ***pages*** *(Integer; required for the first page of a set of separation pages, but not allowed for subsequent pages of the same set) The number of pages which belong to the same set of separation pages comprising the color data for a single composite page. All pages in the set must appear sequentially in the file.* ***spotname*** *(String; required unless spotcolor has been supplied) The name of the colorant for the current page.* ***spotcolor*** *(Spot color handle) A color handle describing the colorant for the current page.*
taborder	*(Keyword; PDF 1.5) Keyword specifying the tab order for form fields and annotations on the page (Default:* `none`*):* ***column*** *Annotations are visited column by column from top to bottom, where columns are ordered as specified by the* `direction` *suboption of the* `viewerpreferences` *option of PDF_begin/end_document( ).* ***none*** *The tab order is unspecified.* ***structure*** *Annotations are visited in the order in which they appear in the structure tree. The oder for annotations that are not included in the structure tree is unspecified.* ***row*** *Annotations are visited row by row starting at the topmost row, where the direction within a row is as specified by the* `direction` *suboption of the* `viewerpreferences` *option of PDF_begin/end_document( ).*
topdown[1]	*(Boolean) If* `true`*, the origin of the coordinate system at the beginning of the page will be assumed in the top left corner of the page, and y coordinates will increase downwards; otherwise the default coordinate system will be used. Default:* `false`

Table 2.9 Options for PDF_begin_page_ext( ) and PDF_end_page_ext( )

option	description
transition	*(Keyword) Set the page transition for the current page in order to achieve special effects which may be useful when displaying the PDF in Acrobat's fullscreen mode as presentations if openmode=fullscreen (see Table 2.3). Default:* `replace`
	split *Two lines sweeping across the screen reveal the page*
	blinds *Multiple lines sweeping across the screen reveal the page*
	box *A box reveals the page*
	wipe *A single line sweeping across the screen reveals the page*
	dissolve *The old page dissolves to reveal the page*
	glitter *The dissolve effect moves from one screen edge to another*
	replace *The old page is simply replaced by the new page*
	fly *(PDF 1.5) The new page flies into the old page.*
	push *(PDF 1.5) The new page pushes the old page off the screen*
	cover *(PDF 1.5) The new page slides on to the screen and covers the old page.*
	uncover *(PDF 1.5) The old page slides off the screen and uncovers the new page.*
	fade *(PDF 1.5) The new page gradually becomes visible through the old one.*
trimbox	*(Rectangle) Change the page box parameters of the current page. The coordinates of the respective box are specified in the default coordinate system. By default, only the MediaBox will be created by using the* `width` *and* `height` *parameters.*
userunit	*(Float or keyword; PDF 1.6) A number in the range 1..75 000 specifying the size of a user unit in points, or one of the keywords mm, cm, m which scales to the respective unit. Default: 1 (i.e. one unit is one point)*
width	*(Float or keyword; not allowed if the* `topdown` *option or parameter is* `true`*) See* `height` *option above.*

1. Only for PDF_begin_page_ext( )

C++ Java ***void suspend_page(String optlist)***
Perl PHP ***PDF_suspend_page(resource p, string optlist)***
C ***void PDF_suspend_page(PDF *p, const char *optlist)***

Suspend the current page so that it can later be resumed.

optlist An option list for future use.

Details The full state of the current page (graphics, color, text, etc.) will be saved internally. It can later be resumed with *PDF_resume_page( )* to add more content. Suspended pages must be resumed before they can be closed.

Scope *page;* this function starts *document* scope, and must always be paired with a matching *PDF_resume_page( )* call. This function must not be used in Tagged PDF mode.

C++ Java ***void resume_page(String optlist)***
Perl PHP ***PDF_resume_page(resource p, string optlist)***
C ***void PDF_resume_page(PDF *p, const char *optlist)***

Resume a page to add more content to it.

optlist An option list according to Table 2.10. The following options can be used: *group, pagenumber*

Details The page must have been suspended with *PDF_suspend_page( )*. It will be opened again so that more content can be added. All suspended pages must be resumed before they can be closed, even if no more content has been added.

Scope *document;* this function starts *page* scope, and must always be paired with a matching *PDF_suspend_page( )* call.

Table 2.10 Options for PDF_resume_page( )

option	***description***
group	*(String; required if the document uses page groups, but not allowed otherwise) Name of the page group of the resumed page. The group name must have been defined with the* `groups` *option in PDF_begin_document( ).*
pagenumber	*(Integer) If this option is supplied, the page with the specified number within the page group chosen in the* `group` *option (or in the document if the document doesn't use page groups) will be resumed. If this option is missing the last page in the group will be resumed.*

2.5 PDFlib Virtual File System (PVF)

C++ Java ***void create_pvf(String filename, byte[] data, String optlist)***

Perl PHP ***PDF_create_pvf(resource p, string filename, string data, string optlist)***

C ***void PDF_create_pvf(PDF *p,***
const char *filename, int len, const void *data, size_t size, const char *optlist)

Create a named virtual read-only file from data provided in memory.

filename (Name string) The name of the virtual file. This is an arbitrary string which can later be used to refer to the virtual file in other PDFlib calls.

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

data A reference to the data for the virtual file. In C and C++ this is a pointer to a memory location. In Java this is a byte array. In Perl, Python, and PHP this is a string.

size (C and C++ only) The length in bytes of the memory block containing the data.

optlist An option list according to Table 2.11. The following options can be used: *copy*

Details The virtual file name can be supplied to any API function which uses input files (virtual files cannot be used for the generated PDF output; use an empty file name in *PDF_begin_document( )* to achieve this). Some of these functions may set a lock on the virtual file until the data is no longer needed. Virtual files will be kept in memory until they are deleted explicitly with *PDF_delete_pvf( ),* or automatically in *PDF_delete( )*.

Each PDFlib object will maintain its own set of PVF files. Virtual files cannot be shared among different PDFlib objects, but they can be used for creating multiple documents with the same PDFlib object. Multiple threads working with separate PDFlib objects do not need to synchronize PVF use. If *filename* refers to an existing virtual file an exception will be thrown. This function does not check whether *filename* is already in use for a regular disk file.

Unless the *copy* option has been supplied, the caller must not modify or free (delete) the supplied data before a corresponding successful call to *PDF_delete_pvf( )*. Not obeying to this rule will most likely result in a crash.

Scope *any*

Table 2.11 Options for PDF_create_pvf( )

option	description
copy	*(Boolean) PDFlib will immediately create an internal copy of the supplied data. In this case the caller may dispose of the supplied data immediately after this call. The* `copy` *option will automatically be set to* `true` *in the COM, .NET, and Java bindings (default for other bindings: false). In other language bindings the data will not be copied unless the* `copy` *option is supplied.*

C++ Java ***int delete_pvf(String filename)***

Perl PHP ***int PDF_delete_pvf(resource p, string filename)***

C ***int PDF_delete_pvf(PDF *p, const char *filename, int len)***

Delete a named virtual file and free its data structures (but not the contents).

filename (Name string) The name of the virtual file as supplied to *PDF_create_pvf( )*.

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

Returns -1 (in PHP: 0) if the corresponding virtual file exists but is locked, and 1 otherwise.

Details If the file isn't locked, PDFlib will immediately delete the data structures associated with *filename*. If *filename* does not refer to a valid virtual file this function will silently do nothing. After successfully calling this function *filename* may be reused. All virtual files will automatically be deleted in *PDF_delete( )*.

The detailed semantics depend on whether or not the *copy* option has been supplied to the corresponding call to *PDF_create_pvf( ):* If the *copy* option has been supplied, both the administrative data structures for the file and the actual file contents (data) will be freed; otherwise, the contents will not be freed, since the client is supposed to do so.

Scope *any*

2.6 Exception Handling

Table 2.12 lists relevant parameter key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 2.12 Exception-related keys for PDF_get/set_parameter( )

key	explanation
errorpolicy	*(Keyword) Controls the behavior of various functions in case of an error. This setting can be overridden by the* `errorpolicy` *option of many functions, and serves as default for the option with the same name:* **legacy** *The behavior of the functions is the same as in PDFlib 6 (controlled by various* `*warning` *parameters and options which are deprecated in PDFlib 7). This setting provides compatibility of existing applications with previous versions of PDFlib.* **return** *If an error occurs the function will return. Functions which can return an error code (e.g. PDF_load_image( )) will return -1 (in PHP: 0). Application developers must check the return value against-1 (in PHP: 0) in order to detect error situations. In case of an error a detailed description of the problem can be queried with PDF_get_errmsg( ). Various* `*warning` *parameters and options will be ignored. This setting is recommended for new applications, and for bringing existing applications up to date.* **exception** *If an error occurs, the function will throw an exception. The (partial) PDF output document will be unusable.* *Default:* `legacy`. *Scope: any*
warning	*Deprecated; warnings no longer trigger exceptions, but are only accessible via logging (see Section 2.7, »Logging«, page 32)*

C++ Java ***int get_errnum( )***
Perl PHP ***int PDF_get_errnum(resource p)***
C ***int PDF_get_errnum(PDF *p)***

Get the number of the last thrown exception or the reason for a failed function call.

Returns The number of an exception, or the reason code of the most recently called function which failed with an error code.

Scope Between an exception thrown by PDFlib and the death of the PDFlib object. Alternatively, this function may be called after a function returned a -1 (in PHP: 0) error code, but before calling any other function except those listed in this section.

Bindings In C++, Java, and PHP 5 this function is also available as *get_errnum( )* in the *PDFlibException* object.

C++ Java ***String get_errmsg( )***
Perl PHP ***string PDF_get_errmsg(resource p)***
C ***const char *PDF_get_errmsg(PDF *p)***

Get the text of the last thrown exception or the reason for a failed function call.

Returns Text containing the description of the last exception thrown, or the reason why the most recently called function failed with an error code.

Scope Between an exception thrown by PDFlib and the death of the PDFlib object. Alternatively, this function may be called after a function returned a -1 (in PHP: 0) error code, but before calling any other function except those listed in this section.

Bindings In C++, Java, and PHP 5 this function is also available as *get_errmsg( )* in the *PDFlibException* object.

C++ Java ***String get_apiname( )***

Perl PHP ***string PDF_get_apiname(resource p)***

C ***const char *PDF_get_apiname(PDF *p)***

Get the name of the API function which threw the last exception or failed.

Returns The name of the function which threw an exception, or the name of the most recently called function which failed with an error code.

Scope Between an exception thrown by PDFlib and the death of the PDFlib object. Alternatively, this function may be called after a function returned a -1 (in PHP: 0) error code, but before calling any other function except those listed in this section.

Bindings In C++, Java, and PHP 5 this function is also available as *get_apiname( )* in the *PDFlibException* object.

C++ ***void *get_opaque( )***

C ***void *PDF_get_opaque(PDF *p)***

Fetch the opaque application pointer stored in PDFlib.

Returns The opaque application pointer stored in PDFlib which has been supplied in the call to *PDF_new2( )*.

Details PDFlib never touches the opaque pointer, but supplies it unchanged to the client. This may be used in multi-threaded applications for storing private thread-specific data within the PDFlib object. It is especially useful for thread-specific exception handling.

Scope *any*

Bindings Only available in the C and C++ bindings.

2.7 Logging

The logging feature can be used to trace API calls. The contents of the log file may be useful for debugging purposes, or may be requested by PDFlib GmbH support. Table 2.13 lists the parameter key names for the logging feature.

Table 2.13 Keys for PDF_set_parameter( ) (see Section 2.1, »Parameter Handling«, page 11)

key	explanation
logging	*Option list with logging options according to Table 2.14*
logmsg	*String which will be copied to the log file*

The *logging* parameter accepts a boolean value or an option list according to Table 2.14. The options can be supplied in the following ways:

- As an option list for the *logging* option of *PDF_set_parameter( )*, e.g.:

```
p.set_parameter("logging", "filename=debug.log remove");
```

- In an environment variable called *PDFLIBLOGGING*. Doing so will activate the log output starting with the very first call to one of the API functions.

Table 2.14 Options for the `logging` *parameter*

key	explanation
(empty list)	*Enable log output*
disable	*(Boolean) Disable logging output*
enable	*(Boolean) Enable logging output*
filename	*(String) Name of the log file (*`stdout` *and* `stderr` *will be recognized as special names). Output will be appended to any existing contents. Default:* *pdflog on MVS* *PDFlib.log on Mac and iSeries* *\PDFlib.log on Windows* */tmp/PDFlib.log on all other systems* *The log file name can alternatively be supplied in an environment variable called PDFLIBLOGFILE.*
flush	*(Boolean) If* `true`*, the log file will be closed after each output, and reopened for the next output to make sure that the output will actually be flushed. This may be useful when chasing program crashes where the log file is truncated, but significantly slows down processing. If* `false`*, the log file will be opened only once. Default:* `true`
remove	*(Boolean) If* `true`*, an existing log file will be deleted before writing new output. Default:* `false`
stringlimit	*(Integer) Limit for the number of characters per line, or 0 for unlimited. Default: 0*

Table 2.14 Options for the `logging` *parameter*

key	***explanation***
classes	*(Option list) List containing options of type integer, where each option describes a logging class and the corresponding value describes the granularity level. Level 0 disables a logging class, positive numbers enable a class. Increasing levels provide more and more detailed output. Default:* `{api=1 warning=1}`.
api	*Log all API calls with their function parameters and results. If* `level=2` *a timestamp will be created in front of all API trace lines.*
filesearch	*Log all attempts related to locating files via* `SearchPath` *or PVF.*
resource	*Log all attempts at locating resources via Windows registry, UPR definitions as well as the results of the resource search.*
user	*User-specified logging output*
warning	*Log all PDFlib warnings, i.e. error conditions which can be ignored or fixed internally.* `If level=2` *messages from functions which do not throw any exception, but hook up the message text for retrieval via PDF_get_errmsg( ), and the reason for all failed attempts at opening a file (searching for a file in searchpath) will also be logged.*

3 Text Functions

3.1 Font Handling

Table 3.1 and Table 3.2 list relevant parameter and value key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 3.1 Font-related keys for PDF_get/set_parameter( )

key	*explanation*
Encoding ***FontAFM*** ***FontPFM*** ***FontOutline*** ***HostFont***	*The corresponding resource file line as it would appear for the respective category in a UPR file. Multiple calls add new entries to the internal list. (See also resourcefile in Table 2.1). Scope: any*
fontwarning	*Deprecated, use* `errorpolicy`
ascenderfaked, capheightfaked, descenderfaked, fontencoding, fontname, fontstyle, xheightfaked *Deprecated, use PDF_info_font( )*	
autocidfont, autosubsetting, unicodemap: Deprecated, use the corresponding option in PDF_load_font( )	

Table 3.2 Font-related keys for PDF_get/set_value( )

key	*explanation*
fontmaxcode, capheight, ascender, descender, xheight, monospace: Deprecated, use PDF_info_font( )	
subsetlimit, subsetminsize: Deprecated, use the corresponding option in PDF_load_font( )	

C++ Java ***int load_font(String fontname, String encoding, String optlist)***

Perl PHP ***int PDF_load_font(resource p, string fontname, string encoding, string optlist)***

C ***int PDF_load_font(PDF *p, const char *fontname, int len, const char *encoding, const char *optlist)***

Search for a font and prepare it for later use.

fontname (Name string) The real or alias name of the font. It will be used to find font data. Case is significant.

len (C language binding only) Length of *fontname* in bytes for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

encoding The encoding to be used with the font (case is significant):

- *unicode* for Unicode-based addressing;
- one of the predefined 8-bit encodings *winansi, macroman, macroman_apple, ebcdic, ebcdic_37, pdfdoc, iso8859-X, cpXXXX*, or *U+XXXX*;
- *host* or *auto* for an automatically selected encoding;
- the name of a user-defined encoding loaded from file or defined via *PDF_encoding_set_char( );*
- *cp932, cp936, cp949,* or *cp950* for CJK codepages (on Windows the system code pages will be used; on all other systems the corresponding CMaps must be available).

- *glyphid for* glyph id addressing;
- *builtin* to select the font's internal encoding (mostly for symbolic fonts);
- the name of a standard CMap;
- *Identity-H* or *Identity-V* for CID addressing with standard CJK fonts and OpenType CID fonts; 2 bytes in native byte ordering must be supplied per glyph; these encodings are mainly useful for creating CJK character collection tables;
- an encoding name known to the operating system (not available on all platforms).

The encoding must be compatible with the chosen font. Table 3.3 details the allowed combinations of encodings and font types.

Table 3.3 Allowed encodings for different font types

font format	*unicode, cp936 etc.*	*8-bit encodings*	*CMaps*	*builtin*	*glyphid*
PostScript Type 1	*yes*[1]	*yes*	–	*Symbol fonts only*	–
Type 3	*yes*[1]	*yes*	–	–	–
TrueType and OpenType with TrueType outlines	*yes*[2]	*yes*[2]	*yes*[2,3]	*Symbol fonts only*	*yes*[2]
Western OpenType with PostScript outlines (SID)	*yes*[2]	*yes*[2]	*yes*[2]	*yes*	*yes*[2]
CJK OpenType with PostScript outlines (CID)	*yes*	–	*yes*	–	*yes*
Standard CJK fonts (no embedding)	*yes*[4]	–	*yes*[5]	–	–

1. A maximum of 256 glyphs can be addressed.
2. PDFlib will force font embedding (exceptions for 8-bit encodings: the encoding contains only characters from the Adobe Standard Latin character set, or `autocidfont=false`*). The font therefore must allow embedding.*
3. The font cannot be used in form fields.
4. A suitable UCS-2 or UTF-16 CMap will be selected based on the character collection for the font, the `vertical` *option, and the current PDF compatibility setting.*
5. A standard CJK font with non-Unicode CMap can be used in Textflow if `keepnative=false`*, and can be used in form fields if* `keepnative=true`*. A standard CJK font can never be used both in Textflow and form fields.*

optlist An option list according to Table 3.4. The following options can be used: *autocidfont, autosubsetting, capheight, descender, embedding, errorpolicy, fontstyle, keepnative, kerning, linegap, metadata, monospace, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*

Returns A font handle for later use with *PDF_info_font( )*, text output functions, and the *font* option of various functions. By default, this function returns an error code of -1 (in PHP: 0) if the requested font/encoding combination cannot be loaded, and does not throw an exception. However, this behavior can be changed with the *errorpolicy* parameter or option. If the function returns -1 (in PHP: 0) you can request the reason of the failure with *PDF_get_errmsg( )*.

The returned number doesn't have any significance to the user other than serving as a font handle. For example, requesting the same font/encoding combination in different documents may result in different font handles.

Details This function prepares a font for later use. The metrics will be loaded from memory, from the system (for host fonts), or from a (virtual or disk-based) metrics file. If the requested font/encoding combination cannot be used due to a configuration problem (e.g. a font, metrics, or encoding file could not be found, or a mismatch was detected), an

error code will be returned or an exception raised (subject to *errorpolicy)*. Otherwise, the value returned by this function can be used as font handle when calling other font-related functions.

When calling this function again with the same font name the same font handle as in the first call will be returned unless a different *encoding* parameter has been supplied, or one of the *fontstyle*, *monospace*, or *vertical* options is different.

Conflicting options: when a font is loaded via *PDF_load_font( )* or requested via *PDF_fill_textblock( )* without embedding, kerning, or subsetting, these options will be ignored if the same font is loaded again later.

Scope *font, document, page, path, pattern, template, glyph*

Params See Table 3.1 and Table 3.2

Table 3.4 Options for PDF_load_font( )

option	*description*
ascender	*(Integer between -2048 and 2048) Force the corresponding typographic property to the specified value. This will override any values found in the font, and is especially useful if the font does not contain any such information (e.g. Type 3 fonts). Default: the value in the font if present, or an estimated value otherwise (which can be queried with PDF_info_font( ))*
autocidfont	*(Boolean) If* `true`*, TrueType fonts with 8-bit encoding except* `winansi`*,* `macroman`*,* `builtin`*, and OpenType fonts without glyph names will automatically be stored as CID fonts. This avoids problems with certain non-accessible glyphs outside* `winansi` *encoding. Default:* `true`
autosubsetting	*(Boolean) Dynamically decide whether or not the font will be subset, subject to the* `subsetlimit` *and* `subsetminsize` *parameters and the actual usage of glyphs. This option will be ignored if the* `subsetting` *option has been supplied. Default:* `true`
capheight	*(Integer between -2048 and 2048) See* `ascender` *above.*
descender	*(Integer between -2048 and 2048) See* `ascender` *above.*
embedding	*(Boolean; must be true for PDF/A and PDF/X) Controls whether or not the font will be embedded. If a font is to be embedded, the font outline file must be available in addition to the metrics information (this is irrelevant for TrueType and OpenType fonts), and the actual font outline definition will be included in the PDF output. If a font is not embedded, only general information about the font is included in the PDF output. Default:* `false` *Font embedding will be enforced for certain font/encoding combinations (see Table 3.3). This option does not have any effect on Type 3 fonts.*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
fontstyle	*(Keyword) Controls the creation of artificial font styles. Possible keywords are* `normal`*,* `bold`*,* `italic`*,* `bolditalic`*. For TrueType (not TTC) and OpenType fonts which are not embedded the artificial font style will be created by Acrobat, otherwise by PDFlib (using the same emboldening method as in the* `fakebold` *option). In the latter case the slanting angle can be controlled with the* `italicangle` *parameter. If this option is applied to one of the core fonts, the appropriate bold, italic, or bolditalic font variant will be chosen instead of creating an artificial font style. If no such font is available (e.g. applying* `bold` *to Times-Bold), the option will be ignored. Default:* `normal`
fontwarning	*Deprecated, use* `errorpolicy`

Table 3.4 Options for PDF_load_font( )

option	description
keepnative	*(Boolean; only relevant for standard CJK fonts with a non-Unicode CMap; will be ignored for other fonts) If* `false`*, text in this font will be converted to Unicode (using* `glyphid` *addressing and* `Identity-H` *encoding) when creating PDF output. This does not affect the text supplied to API functions which must still match the selected CMap (e.g. Shift-JIS) or the visible output. However, the font can be used in Textflow and all simple text output functions (but not in form fields).* *If true, text in this font will be written to the PDF output in its native format according to the specified CMap. The font can be used in form fields and all simple text output functions, but not in Textflow. Default:* `true`
kerning	*(Boolean) Controls whether or not kerning values will be read from the font. Default:* `false`
linegap	*(Integer between -2048 and 2048) See* `ascender` *above.*
metadata	*(Option list; PDF 1.4) Supply metadata for the font (see Section 12.2, »XMP Metadata«, page 161)*
monospace	*(Integer between 1 and 2048; not for PDF/A) Forces all glyphs in the font to use the specified width (in the font coordinate system: 1000 units equal the font size). For Type 3 fonts all glyph widths which are different from 0 will be modified. This option is only recommended for standard CJK fonts, and not supported for core fonts; it will be ignored if the font is embedded. Default: absent (metrics from the font will be used)*
replacementchar	*(Unichar; only relevant if* `glyphcheck=replace`*) Glyphs which are not available in the selected font and which cannot be substituted will be replaced with the specified Unicode value (or the specified code in case of* `builtin` *encoding). U+0000 can be used to specify the font's »missing glyph« symbol. Default: no replacement character*
subsetlimit	*(Float or percentage; will be ignored for Type 3 fonts) Automatic font subsetting will be disabled if the percentage of glyphs used in the document related to the total number of glyphs in the font exceeds the provided percentage. Default: 100%*
subsetminsize	*(Float; will be ignored for Type 3 fonts) Automatic font subsetting will be disabled if the size of the original font file is less than the provided value in KB. Default: 50*
subsetting	*(Boolean) Controls whether or not the font will be subset. Subsetting for Type 3 fonts requires a two-pass definition of the font, and the* `subsetting` *option must be provided in the first call to PDF_load_font( ). Default:* `false`
unicodemap	*(Boolean; must not be set to false for pdfa=PDF/A-1a:2005) Controls the generation of ToUnicode CMaps. This option will be ignored in Tagged PDF mode. Default:* `true`
vertical	*(Boolean; only for TrueType and OpenType fonts; will be ignored if a predefined CMap is specified, and will be forced to true if the font name starts with @) If* `true`*, the font will be prepared for vertical writing mode.*
xheight	*(Integer between -2048 and 2048) See* `ascender` *above.*

C++ Java ***double info_font(int font, String keyword, String optlist)***

Perl PHP ***float PDF_info_font(resource p, int font, string keyword, string optlist)***

C ***double PDF_info_font(PDF *p, int font, const char *keyword, const char *optlist)***

Query detailed information about a loaded font.

font A font handle returned by *PDF_load_font( )*.

keyword A keyword specifying the requested information according to Table 3.5.

optlist An option list according to Table 3.5. The following options can be used:

ascender, capheight, cidfont, code, descender, encoding, fontfile, fontname, fontstyle, glyphid, glyphname, hostfont, italicangle, kerningpairs, linegap, maxcode, metricsfile, numcids, numglyphs, replacementchar, standardfont, supplement, symbolfont, unicode, unicodefont, vertical, weight, willembed, willsubset, xheight

Returns The value of some font property as requested by *keyword* and in some cases auxiliary options. For unspecified combinations of keyword and options -1 will be returned. Some keywords will return a string indirectly by returning its string index. The corresponding string can be retrieved via *PDF_get_parameter( )* and the *string* parameter (see Table 2.1).

Scope any except *object*

Table 3.5 Keywords and options for PDF_info_font( )

keyword	***explanation***
ascender	*Metrics value for the ascender. Supported options (default:* `fontsize=1000`*):* ***faked*** *(Boolean) 1 if the value had to be estimated because it was not available in the font or metrics file, otherwise 0* ***fontsize*** *(Float) Value will be scaled to the specified font size*
capheight	*Metrics value for the capheight. See* `ascender`.
cidfont	*1 if the font will be embedded as a CID font, otherwise 0*
code	*(Only for fonts with 8-bit encoding) Number in the range 0...255 specifying an encoding slot which specifies a condition according to one of the following options, or -1 if no such slot could be found:* ***unicode*** *(Unichar) Unicode character* ***glyphname*** *(String) Slot with the specified glyph name*
descender	*Metrics value for the descender. See* `ascender`.
encoding	*String index of the name of the font's encoding or CMap. Supported options (default:* `actual`*):* ***api*** *(Boolean) If* `true`*, the encoding name as specified in the API* ***actual*** *(Boolean) If* `true`*, the name of the actual encoding used for the font*
fontfile	*String index of the path name for the font outline file, or -1 if unavailable*
fontname	*String index of the font name, or -1 if unavailable. Supported options (default:* `acrobat`*):* ***api*** *(Boolean) If* `true`*, the font name as specified in the API* ***full*** *(Boolean) If* `true`*, the /FontName entry in the PDF font descriptor* ***acrobat*** *(Boolean) If* `true`*, the font name as displayed in Acrobat*
fontstyle	*String index for the value of the* `fontstyle` *option* (`normal`, `bold`, `italic`, *or* `bolditalic`). *Supported option:* ***faked*** *1 if fontstyle will be realized by PDFlib, 0 if fontstyle will be realized by Acrobat*
glyphid	*(For fonts with 8-bit encoding, Symbol fonts with* `encoding=builtin`*, and fonts with* `encoding=unicode`*) Number in the range 0...65535 specifying the font-internal glyph id (GID) specified with the following option, or -1 if no such glyph could be found:* ***code*** *(Number in the range 0...255; only for fonts with 8-bit encoding and Symbol fonts with* `encoding=builtin`*) Encoding slot* ***unicode*** *(Unichar; only for fonts with* `encoding=unicode`*) Slot with the specified Unicode character*

Table 3.5 Keywords and options for PDF_info_font( )

keyword	explanation
glyphname	*(For all fonts except Symbol fonts with* `encoding=builtin` *and fonts with a standard CMap) String index of the name of the glyph specified by one of the following options, or -1 if no such glyph could be found:* ***code*** *(Number in the range 0...255; only for fonts with 8-bit encoding) Encoding slot* ***glyphid*** *(Number in the range 0...65535; only for TT/OT fonts with* `encoding=unicode` *and fonts with* `encoding=glyphid`*) Internal glyph id* ***unicode***[1] *(Unichar) Slot with the specified Unicode character*
hostfont	*1 if the font is a host font, 0 otherwise*
italicangle	*Italic angle of the font (/ItalicAngle in the PDF font descriptor)*
kerningpairs	*Number of kerning pairs in the font*
linegap	*Metrics value for the linegap. See* `ascender`.
maxcode	*Highest code value for the font's encoding*
metricsfile	*String index of the path name for the font metrics file (AFM or PFM), or -1 if unavailable*
numcids	*Number of CIDs if font uses a standard CMap, otherwise -1*
numglyphs	*Number of glyphs in the font*
replacement char	*Unicode value (for Unicode-compatible fonts) or code value (for Symbol fonts) of the replacement character of the font*
standardfont	*1 if the font is a PDF core font or a standard CJK font, otherwise 0*
supplement	*Supplement number of the character collection for fonts with a standard CJK CMap, otherwise 0*
symbolfont	*1 if the font is a symbolic font, 0 otherwise (symbol flag in the PDF font descriptor; decision is based on font data)*
unicode	*(For all fonts except Symbol fonts with* `encoding=builtin`*) Unicode UTF-32 value for the glyph specified by one of the following options, or -1 if no Unicode value could be found:* ***cid*** *(Number; only for fonts with a standard CMap) CID value of the glyph* ***code*** *(Number in the range 0...255; only for fonts with 8-bit encoding) Encoding slot* ***glyphid*** *(Number in the range 0...65535; only for TT/OT fonts with* `encoding=unicode` *and fonts with* `encoding=glyphid`*) Internal glyph id* ***glyphname***[1] *(String; only for fonts with 8-bit encoding) Name of the glyph for which to determine a Unicode value*
unicodefont	*1 if the font was loaded with an encoding that allows Unicode text (*`builtin`*,* `glyphid`*, and non-Unicode CMaps do not allow Unicode text), otherwise 0*
vertical	*1 if the font is for vertical writing mode, otherwise 0*
weight	*Font weight in the range 100...900; 400=normal, 700=bold*
willembed	*1 if the font will be embedded (via the* `embedding` *option or forced font embedding), otherwise 0*
willsubset	*1 if a font subset will be created (if* `autosubsetting=true`*, the* `subsetlimit` *must be reached for subsetting to be activated), otherwise 0*
xheight	*Metrics value for the xheight. See* `ascender`.

1. The keyword `glyphname` *and option* `unicode` *(and vice versa) can be used to determine the relationship of glyph names and Unicode values for fonts with an 8-bit encoding. However, this combination can also be used to determine these mappings based on PDFlib's internal mapping tables independently from any specific 8-bit encoding. For this scenario a font handle for a font without any 8-bit encoding must be supplied.*

3.2 Type 3 Font Definition

C++ Java ***void begin_font(String fontname,***
double a, double b, double c, double d, double e, double f, String optlist)

Perl PHP ***PDF_begin_font(resource p, string fontname,***
float a, float b, float c, float d, float e, float f, string optlist)

C ***void PDF_begin_font(PDF *p, char *fontname, int reserved,***
double a, double b, double c, double d, double e, double f, const char *optlist)

Start a Type 3 font definition.

fontname (Name string) The name under which the font will be registered, and can later be used with *PDF_load_font( )*.

reserved (C language binding only) Reserved, must be 0.

a, b, c, d, e, f (Will be ignored in the second pass for subset fonts) The elements of the font matrix. This matrix defines the coordinate system in which the glyphs will be drawn. The six values make up a matrix in the same way as in PostScript and PDF (see references). In order to avoid degenerate transformations, *a*d* must not be equal to *b*c*. A typical font matrix for a 1000 *x* 1000 coordinate system is *[0.001, 0, 0, 0.001, 0, 0]*.

optlist (Ignored in the second pass for subset fonts) An option list according to Table 3.6. The following options can be used: *colorized, familyname, stretch, weight, widthsonly*

Details This function will reset all text, graphics, and color state parameters to their defaults. The font may contain an arbitrary number of glyphs, but only 256 glyphs can be accessed via an encoding. The font can be used until the end of the current *document* scope.

Scope *document, page;* this function starts *font* scope, and must always be paired with a matching *PDF_end_font( )* call. For the second pass of subsetted fonts only *document* scope is allowed.

Table 3.6 Options for PDF_begin_font( )

option	*description*
colorized	*(Boolean) If* `true`*, the font may explicitly specify the color of individual characters. If* `false`*, all characters will be drawn with the current color (at the time the font is used, not when it is defined), and the glyph definitions must not contain any color operators or images other than masks. Default:* `false`
familyname[1]	*(String; PDF 1.5) Name of the font family*
stretch[1]	*(Keyword; PDF 1.5) The font stretch value. Keywords:* `ultracondensed`*,* `extracondensed`*,* `condensed`*,* `semicondensed`*,* `normal`*,* `semiexpanded`*,* `expanded`*,* `extraexpanded`*,* `ultraexpanded`*. Default:* `normal`
weight[1]	*(Integer or keyword; PDF 1.5) The font weight. Numbers and equivalent keywords: 100=*`thin`*, 200=*`extralight`*, 300=*`light`*, 400=*`normal`*, 500=*`medium`*, 600=*`semibold`*, 700=*`bold`*, 800=*`extrabold`*, 900=*`black`*. Default:* `normal`
widthsonly	*(Boolean) If* `true`*, only the metrics of the font and glyphs will be defined. API function calls between PDF_begin_glyph( ) and PDF_end_glyph( ) will be ignored. The actual glyph outlines must be defined in a separate font scope and loop over all glyphs. This enables PDFlib to perform subsetting on Type 3 fonts; because glyph definitions will be ignored in the first pass client applications can supply the same glyph definitions in both passes. Default:* `false`

1. These options are strongly recommended when creating Tagged PDF, and will be ignored otherwise.

C++ Java ***void end_font( )***
Perl PHP ***PDF_end_font(resource p)***
C ***void PDF_end_font(PDF *p)***

Terminate a Type 3 font definition.

Scope *font;* this function terminates *font* scope, and must always be paired with a matching *PDF_begin_font( )* call.

C++ Java ***void begin_glyph(String glyphname, double wx, double llx, double lly, double urx, double ury)***
Perl PHP ***PDF_begin_glyph(resource p, string glyphname, float wx, float llx, float lly, float urx, float ury)***
C ***void PDF_begin_glyph(PDF *p,***
char *glyphname, double wx, double llx, double lly, double urx, double ury)

Start a glyph definition for a Type 3 font.

glyphname The name of the glyph. This name must be used in any encoding which will be used with the font. Glyph names within a font must be unique.

wx (Will be ignored in the second pass for subset fonts) The width of the glyph in the glyph coordinate system, as specified by the font's matrix.

llx, lly, urx, ury (Will be ignored in the second pass for subset fonts) If the font's *colorized* option is false (which is default), the coordinates of the lower left and upper right corners of the glyph's bounding box. The bounding box values must be correct in order to avoid problems with PostScript printing. If the font's *colorized* option is *true*, all four values must be 0.

Details The glyphs in a font can be defined using text, graphics, and image functions. Images, however, can only be used if the font's *colorized* option is *true*, or the image has been opened with the *mask* option. It is strongly suggested to use the inline image feature for defining bitmaps in Type 3 fonts.

Since the complete graphics state of the surrounding page will be inherited for the glyph definition when the *colorized* option is *true*, the glyph definition should explicitly set any aspect of the graphics state which is relevant for the glyph definition (e.g. *linewidth)*.

Scope *page, font;* this function starts *glyph* scope, and must always be paired with a matching *PDF_end_glyph( )* call. If *widthsonly=true* in *PDF_begin_font( )* all API function calls between *PDF_begin_glyph( )* and *PDF_end_glyph( )* will be ignored.

C++ Java ***void end_glyph( )***
Perl PHP ***PDF_end_glyph(resource p)***
C ***void PDF_end_glyph(PDF *p)***

Terminate a glyph definition for a Type 3 font.

Scope *glyph;* this function terminates *glyph* scope, and must always be paired with a matching *PDF_begin_glyph( )* call.

3.3 Encoding Definition

C++ Java ***void encoding_set_char(String encoding, int slot, String glyphname, int uv)***

Perl PHP ***PDF_encoding_set_char(resource p, string encoding, int slot, string glyphname, int uv)***

C ***void PDF_encoding_set_char(PDF *p, const char *encoding, int slot, const char *glyphname, int uv)***

Add a glyph name and/or Unicode value to a custom 8-bit encoding.

encoding The name of the encoding. This is the name which must be used with *PDF_load_font( )*. The encoding name must be different from any built-in encoding and all previously used encodings.

slot The position of the character to be defined, with *0 <= slot <= 255*. A particular slot must only be filled once within a given encoding.

glyphname The character's name.

uv The character's Unicode value.

Details This function is only required for specialized applications which must work with non-standard 8-bit encodings. It can be called multiply to define up to 256 character slots in an encoding. More characters may be added to a particular encoding until it has been used for the first time; otherwise an exception will be raised. Not all code points must be specified; undefined slots will be filled with *.notdef* and U+0000.

There are three possible combinations of glyph name and Unicode value:

- *glyphname* supplied, *uv=0:* this parallels an encoding file without Unicode values;
- *uv* supplied, but no *glyphname* supplied*:* this parallels a codepage file;
- *glyphname* and *uv* supplied: this parallels an encoding file with Unicode values.

The defined encoding can be used until the end of the current *object* scope.

Scope *object, document, page, pattern, template, path, font, glyph*

3.4 Simple Text Output

Note *All text supplied to the functions in this section must match the encoding selected with PDF_load_font( ) and the specified textformat. Due to restrictions in Acrobat, text strings must not exceed 32 KB in length.*

Table 3.7 and Table 3.8 lists relevant parameters and values for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 3.7 Text-related keys for PDF_get/set_parameter( )

key	explanation
autospace	*If* `true` *and the current font is Unicode-compatible, PDFlib will automatically add a space character (0x20) after each text output generated with a show operation. This may be useful for generating Tagged PDF. Note that adding spaces changes the current text position after the show operation. Default:* `false`. *Scope: any*
charref	*If* `true`, *enable substitution of numeric and character entity references and glyph name references. Default:* `false`
escape-sequence	*If* `true`, *enable substitution of escape sequences in content strings, hypertext strings, and name strings. Default:* `false`
fakebold	*If* `true`, *simulate bold font by triple overprinting. It is strongly recommended to use bold font variations for emphasis; this parameter will create text output which is inferior to real bold text, and may inhibit text extraction. Default:* `false`
glyphcheck	*The default glyph checking policy (see Table 4.1). Default:* `replace`. *Scope: any*
glyphwarning	*Deprecated, use* `errorpolicy`
kerning	*If* `true`, *enable kerning for fonts which have been opened with the* `kerning` *option; disable if* `false`. *Default:* `true`. *Scope: any*
textformat	*(Only for non Unicode-compatible language bindings) The format in which the text output functions will expect the client-supplied strings. Possible values are* `bytes`, `utf8`, `ebcdicutf8` *(only on iSeries and zSeries),* `utf16`, `utf16le`, `utf16be`, *and* `auto`. *Default:* `auto`. *Scope: any*
underline overline strikeout	*The current underline, overline, and strikeout modes, which are retained until they are explicitly changed, or a new page is started. Theses modes can be set independently from each other, and are reset to* `false` *at the beginning of each page. Default: false. Scope: page, pattern, template, glyph* ***true*** *underline/overline/strikeout text* ***false*** *do not underline/overline/strikeout text*

Table 3.8 Text-related keys for PDF_get/set_value( )

key	explanation
charspacing	*Character spacing, i.e. the shift of the current point after placing individual characters in a string. It is specified in units of the user coordinate system, and is reset to the default of 0 at the beginning and end of each page. In order to spread characters apart use positive values for horizontal writing mode, and negative values for vertical writing mode. Scope: page, pattern, template, glyph, document.*
font[1]	*Identifier of the current font which has been set with PDF_setfont( ), or -1 (in PHP: 0) if no font is set. Scope:* `page`, `pattern`, `template`, `glyph`
fontsize[1]	*Size of the current font which must have been previously set with PDF_setfont( ). Scope:* `page`, `pattern`, `template`, `glyph`

Table 3.8 Text-related keys for PDF_get/set_value( )

key	explanation
horizscaling	Horizontal text scaling to the given percentage (must be different from 0). Text scaling shrinks or expands the text by a given percentage. It is set to the default of 100 at the beginning and end of each page. Text scaling always relates to the horizontal coordinate. Scope: page, pattern, template, glyph, document.
italicangle	The italic (slant) angle of text in degrees (between -90° and 90°). Negative values can be used to simulate italic text when only a regular font is available, especially for CJK fonts . Default: 0 (this parameter will be reset at the beginning and end of each page). Scope: page, pattern, template, glyph, document
leading	Leading, which is the distance between baselines of adjacent lines of text. The leading is used for PDF_continue_text( ). It is set to the value of the font size when a new font is selected using PDF_setfont( ). Setting the leading equal to the font size results in dense line spacing (leading = 0 will result in overprinting lines). However, ascenders and descenders of adjacent lines will generally not overlap. Scope: page, pattern, template, glyph
textrendering	Current text rendering mode. It is set to the default of 0 at the beginning of each page. Scope: page, pattern, template, glyph. 0 fill text 1 stroke text (outline) 2 fill and stroke text 3 invisible text 4 fill text and add it to the clipping path 5 stroke text and add it to the clipping path 6 fill and stroke text and add it to the clipping path 7 add text to the clipping path
textrise	Text rise parameter, which specifies the distance between the desired text position and the default baseline. Positive values of text rise move the text up. The text rise always relates to the vertical coordinate. This may be useful for superscripts and subscripts. The text rise is set to the default value of 0 at the beginning of each page. Scope: page, pattern, template, glyph
textx[1] **texty**[1]	x or y coordinate of the current text position. Default: 0. Scope: page, pattern, template, glyph
underline-position	Position of the stroked line for underlined text, relative to the baseline (as a fraction of the font size). The value 1000000 can be supplied to set a font-specific value which will be retrieved from the font metrics or outline file. Default: 1000000
underline-width	Absolute line width for underlined text. The value 0 can be supplied to set a font-specific value which will be retrieved from the font metrics or outline file. Default: 0
wordspacing	Word spacing, i.e. the shift of the current point after placing individual words in a line. In other words, the current point is moved horizontally after each space character (U+0020). The spacing value is given in text space units, and is reset to the default of 0 at the beginning and end of each page. Scope: page, pattern, template, glyph, document

1. Only for PDF_get_value( )

C++ Java ***void PDF_setfont(int font, double fontsize)***

Perl PHP ***PDF_setfont(resource p, int font, float fontsize)***

C ***void PDF_setfont(PDF *p, int font, double fontsize)***

Set the current font in the specified size.

font A font handle returned by *PDF_load_font( )*.

fontsize Size of the font, measured in units of the current user coordinate system. The font size must not be 0; a negative font size will result in mirrored text relative to the current transformation matrix.

Details The font must be set on each page before calling any of the simple text output functions. Font settings will not be retained across pages. The current font can be changed an arbitrary number of times per page. In all text formatting functions (see Chapter 4, »Formatting Functions«, page 53). the font can be specified using the *font* and *fontsize* options.

Scope *page, pattern, template, glyph*

Params This function automatically sets the *leading* parameter to *fontsize*.

C++ Java ***void set_text_pos(double x, double y)***

Perl PHP ***PDF_set_text_pos(resource p, float x, float y)***

C ***void PDF_set_text_pos(PDF *p, double x, double y)***

Set the position for text output on the page.

x, y The current text position to be set.

Details The text position is set to the default value of *(0, 0)* at the beginning of each page. The current point for graphics output and the current text position are maintained separately.

Scope *page, pattern, template, glyph*

Params See Table 3.7 and Table 3.8.

C++ Java ***void show(String text)***

Perl PHP ***PDF_show(resource p, string text)***

C ***void PDF_show(PDF *p, const char *text)***

C ***void PDF_show2(PDF *p, const char *text, int len)***

Print text in the current font and size at the current text position.

text (Content string) The text to be printed. In C *text* must not contain null characters when using *PDF_show( )*, since it is assumed to be null-terminated; use *PDF_show2( )* for strings which may contain null characters.

len (Only for *PDF_show2( )*) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

Details The font must have been set before with *PDF_setfont( )*. The current text position is moved to the end of the printed text.

Scope *page, pattern, template, glyph*

Params See Table 3.7 and Table 3.8.

Bindings *PDF_show2( )* is only available in C since in all other bindings arbitrary string contents can be supplied with *PDF_show( )*.

C++ Java ***void xshow(String text, const double *xadvancelist)***

C ***void PDF_xshow(PDF *p, const char *text, int len, const double *xadvancelist)***

Print text in the current font and size, using individual horizontal positions.

text (Content string) The text to be printed.

len (Only for the C language binding) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

xadvancelist An array of *x* advance values for the glyphs in text. Each value specifies the relative horizontal displacement (in user coordinates) after a glyph has been placed. The array length must be equal to the number of glyphs in text (not necessarily equal to *len*, which is the the number of bytes!).

Details The font must have been set before with *PDF_setfont( )*.

Scope *page, pattern, template, glyph*

Params See Table 3.7 and Table 3.8.

Bindings Only available in the C and C++ language bindings. Other bindings can use the *xadvancelist* option in *PDF_fit_textline( )* to achieve the same functionality.

C++ Java ***void show_xy(String text, double x, double y)***

Perl PHP ***PDF_show_xy(resource p, string text, float x, float y)***

C ***void PDF_show_xy(PDF *p, const char *text, double x, double y)***

C ***void PDF_show_xy2(PDF *p, const char *text, int len, double x, double y)***

Print text in the current font.

text (Content string) The text to be printed. In C *text* must not contain null characters when using *PDF_show_xy( )*, since it is assumed to be null-terminated; use *PDF_show_xy2( )* for strings which may contain null characters.

x, y The position in the user coordinate system where the text will be printed.

len (Only for *PDF_show_xy2( )*) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

Details The font must have been set before with *PDF_setfont( )*. The current text position is moved to the end of the printed text.

Scope *page, pattern, template, glyph*

Params See Table 3.7 and Table 3.8.

Bindings *PDF_show_xy2( )* is only available in C since in all other bindings arbitrary string contents can be supplied with *PDF_show_xy( )*.

C++ Java ***void continue_text(String text)***

Perl PHP ***PDF_continue_text(resource p, string text)***

C ***void PDF_continue_text(PDF *p, const char *text)***

C ***void PDF_continue_text2(PDF *p, const char *text, int len)***

Print text at the next line.

text (Content string) The text to be printed. If this is an empty string, the text position will be moved to the next line anyway. In C *text* must not contain null characters when using *PDF_continue_text( )*, since it is assumed to be null-terminated; use *PDF_continue_text2( )* for strings which may contain null characters.

len (Only for *PDF_continue_text2( )*) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided as in *PDF_continue_text( )*.

Details The positioning of text *(x* and *y* position) and the spacing between lines is determined by the *leading* parameter and the most recent call to *PDF_fit_textline( ), PDF_show_xy( )* or *PDF_set_text_pos( )*. The current point will be moved to the end of the printed text; the *x* position for subsequent calls of this function will not be changed.

Scope *page, pattern, template, glyph;* this function should not be used in vertical writing mode.

Params See Table 3.7 and Table 3.8.

Bindings *PDF_continue_text2( )* is only available in C since in all other bindings arbitrary string contents can be supplied with *PDF_continue_text( )*.

C++ Java ***double stringwidth(String text, int font, double fontsize)***

Perl PHP ***float PDF_stringwidth(resource p, string text, int font, float fontsize)***

C ***double PDF_stringwidth(PDF *p, const char *text, int font, double fontsize)***

C ***double PDF_stringwidth2(PDF *p, const char *text, int len, int font, double fontsize)***

Calculate the width of *text* in an arbitrary font.

text (Content string) The text for which the width will be queried. In C *text* must not contain null characters when using *PDF_stringwidth( )*, since it is assumed to be null-terminated; use *PDF_stringwidth2( )* for strings which may contain null characters.

len (Only for *PDF_stringwidth2( )*) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

font A font handle returned by *PDF_load_font( )*.

fontsize Size of the font, measured in units of the user coordinate system (see *PDF_setfont( )*).

Returns The width of *text* in a font which has been selected with *PDF_load_font( )* and the supplied *fontsize*. The returned width value may be negative (e.g. when negative horizontal scaling has been set). In vertical writing mode the width of the widest glyph will be returned (use *PDF_info_textline( )* to determine the actual height of the text).

Details The width calculation takes the current values of the following text parameters into account: *horizscaling*, *kerning*, *charspacing*, and *wordspacing*.

Scope *font, page, pattern, template, path, glyph, document*

Params See Table 3.7 and Table 3.8.

Bindings *PDF_stringwidth2( )* is only available in C since in all other bindings arbitrary string contents can be supplied with *PDF_stringwidth( )*.

3.5 Unicode Conversion Functions

These functions may be useful for Unicode string conversion. They are provided for the benefit of users working with language environments that are not Unicode-aware.

C++ ***string utf16_to_utf8(string utf16string)***

Perl PHP ***string PDF_utf16_to_utf8(resource p, string utf16string)***

C ***const char *PDF_utf16_to_utf8(PDF *p, const char *utf16string, int len, int *size)***

Convert a string from UTF-16 format to UTF-8.

utf16string The string to be converted. A Byte Order Mark (BOM) in the string will be interpreted. If it is missing the platform's native byte ordering is assumed.

len (C language binding only) Length of *utf16string* in bytes.

size (C language binding only) C-style pointer to a memory location where the length of the returned string (in bytes) will be stored. If the pointer is NULL it will be ignored.

Returns The converted UTF-8 string. The generated UTF-8 string will start with the UTF-8 BOM *(\xEF\xBB\xBF)*. On EBCDIC platforms the conversion result including the BOM will finally be converted to EBCDIC. The returned string is valid until the next call to any PDFlib function, or until an exception is thrown. Clients must copy the string if they need it longer. The memory used for the converted string will be managed by PDFlib.

Scope *any*

Bindings This function is not available in Unicode-capable language bindings.

C++ ***string utf8_to_utf16(string utf8string, string ordering)***

Perl PHP ***string PDF_utf8_to_utf16(resource p, string utf8string, string ordering)***

C ***const char *PDF_utf8_to_utf16(PDF *p, const char *utf8string, const char *ordering, int *size)***

Convert a string from UTF-8 format to UTF-16.

utf8string The string to be converted, which must contain a valid UTF-8 sequence (on EBCDIC platforms it must be encoded in EBCDIC). If a Byte Order Mark (BOM) is present, it will be removed.

ordering Specifies the byte ordering of the result string:

- *utf16* or an empty string: the converted string will not have any BOM, and will be stored in the platform's native byte order.
- *utf16le:* the converted string will be formatted in little endian format, and will be prefixed with the little-endian BOM *(\xFF\xFE)*.
- *utf16be:* the converted string will be formatted in big endian format, and will be prefixed with the big-endian BOM *(\xFE\xFF)*.

size (C language binding only) C-style pointer to a memory location where the length of the returned string (in bytes) will be stored.

Returns The converted UTF-16 string. The returned string is valid until the next call to any PDFlib function, or until an exception is thrown. Clients must copy the string if they need it longer. The memory used for the converted string will be managed by PDFlib.

Scope *any*

Bindings This function is not available in Unicode-capable language bindings.

C++ ***string utf32_to_utf16(string utf32string, string ordering)***

Perl PHP ***string PDF_utf32_to_utf16(resource p, string utf32string, string ordering)***

C ***const char *PDF_utf32_to_utf16(PDF *p, const char *utf32string, int len, const char *ordering, int *size)***

Convert a string from UTF-32 format to UTF-16.

utf32string The string to be converted, which must contain a valid UTF-32 sequence (on EBCDIC platforms it must be encoded in EBCDIC). If a Byte Order Mark (BOM) is present, it will be interpreted

len (C language binding only) Length of *utf32string* in bytes.

ordering Specifies the byte ordering of the result string:

- *utf16* or an empty string: the converted string will not have any BOM, and will be stored in the platform's native byte order.
- *utf16le:* the converted string will be formatted in little endian format, and will be prefixed with the little-endian BOM *(\xFF\xFE)*.
- *utf16be:* the converted string will be formatted in big endian format, and will be prefixed with the big-endian BOM *(\xFE\xFF)*.

size (C language binding only) C-style pointer to a memory location where the length of the returned string (in bytes) will be stored.

Returns The converted UTF-16 string. The returned string is valid until the next call to any PDFlib function other than *PDF_utf16_to_utf8( )*, *PDF_utf8_to_utf16( )*, and *PDF_utf32_to_utf16( )*, or until an exception is thrown. Clients must copy the string if they need it longer. The memory used for the converted string will be managed by PDFlib.

Scope *any*

Bindings This function is not available in Unicode-capable language bindings.

4 Formatting Functions

4.1 Single-Line Text with Textlines

C++ Java ***void fit_textline(String text, double x, double y, String optlist)***

Perl PHP ***PDF_fit_textline(resource p, string text, float x, float y, string optlist)***

C ***void PDF_fit_textline(PDF*p, const char *text, int len, double x, double y, const char *optlist)***

Place a single line of text at position *(x, y)* subject to various options.

text (Content string) The text to be printed.

len (C language binding only) Length of *text* (in bytes) for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

x, y The coordinates of the reference point in the user coordinate system where the text will be placed, subject to various options.

optlist An option list specifying options according to Table 4.1. The following options can be used:

- Font-related options: *encoding, font, fontname, fontsize*
  Both of the options *fontname* and *encoding* (corresponding to the same-named parameters of *PDF_load_font( ))* can be used to select a font. Alternatively, the *font* option can be used to supply a font handle which has been created with an earlier call to *PDF_load_font( )*. If *font* is specified, the *fontname* and *encoding* options will be ignored. The *fontsize* option is required.
- All options for *PDF_load_font( )* (see Table 3.4). These options will only be used if both the *fontname* and *encoding* options are supplied (but not with the *font* option: *ascender, autocidfont, autosubsetting, capheight, descender, embedding, fontstyle, keepnative, kerning, linegap, metadata, monospace, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*
- Formatting: *alignchar, boxsize, fitmethod, leader, margin, orientate, position, rotate, stamp, xadvancelist*
- Appearance: *charref, charspacing, dasharray, escapesequence, fakebold, fillcolor, glyphcheck, horizscaling, italicangle, kerning, matchbox, overline, showborder, shrinklimit, strikeout, strokecolor, strokewidth, textformat, textrendering, textrise, underline, underlineposition, underlinewidth, wordspacing*

Details The current text and graphics state parameters will be used to control the appearance of the text output unless they are explicitly overridden by options. On the other hand, the current text and graphics state will not be modified by this function (in particular, the current font will be unaffected). However, the *textx/texty* parameters will be adjusted to point to the end of the generated text output.

Scope *page, pattern, template, glyph;* this function should not be used in vertical writing mode.

Params See Table 3.7.

Table 4.1 Options for PDF_fit_textline( ) and PDF_info_textline( )

key	explanation
alignchar	*(Unichar or keyword) If the specified character is found in the text, its lower left corner will be aligned at the reference point. For horizontal text with* `orientate=north` *or* `south` *the first value supplied in the* `position` *option defines the position. For horizontal text with* `orientate=west` *or* `east` *the second value supplied in the* `position` *option defines the position. This option will be ignored if the specified alignment character is not present in the text. The value 0 and the keyword* `none` *suppress alignment characters. The specified* `fitmethod` *will be applied, although the text cannot be placed within the fitbox because of the forced positioning of* `alignchar`**.** *Default:* `none`
boxsize	*(List of floats) Two values specifying the width and height of a box, relative to which the text will be placed and possibly scaled. The lower left corner of the box coincides with the reference point (x, y). Placing the text and fitting it into the box is controlled by the* `position` *and* `fitmethod` *options. If* `width=0`*, only the height is considered; If height=0, only the width is considered. In these cases the text will be placed relative to the vertical line from (x, y) to (x, y+height), or the horizontal line from (x, y) to (x+width, y), respectively. Default:* `{0 0}`
charref	*(Boolean) If* `true`*, enable substitution of numeric and character entity references and glyph name references. Default: the global* `charref` *parameter*
charspacing	*(Float or percentage) The character spacing (see Table 3.7). Percentages are based on* `fontsize`*. Default[1] for PDF_add/create_textflow( ): 0*
dasharray	*(List of two floats) The lengths of dashes and gaps for stroked (outline) text and decoration. Default[1] for PDF_add/create_textflow( ):* `{0 0}` *(i.e. a solid line)*
encoding	*(String; must be used with the* `fontname` *option; will be ignored if* `font` *is specified) Encoding name*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
escape-sequence	*(Boolean) If* `true`*, enable substitution of escape sequences in content strings, hypertext strings, and name strings. Default: the global* `escapesequence` *parameter*
fakebold	*If* `true`*, simulate bold font by triple overprinting. It is strongly recommended to use bold font variations for emphasis; this option will create text output which is inferior to real bold text, and may inhibit text extraction. Default[1]:* `false`
fillcolor	*(Color) Fill color of the text. Default for PDF_fit_textline( ): the corresponding parameter in the current graphics state. Default for PDF_add/create_textflow( ):* `{gray 0}`
fitmethod	*(Keyword) Specifies the method used to fit the text into the specified box. This option will be ignored if no box has been specified. Default:* `nofit`**.** **nofit** — *Position the text only, without any scaling or clipping.* **clip** — *Position the text, and clip it at the edges of the box.* **meet** — *Position the text according to the* `position` *option, and scale it such that it entirely fits into the box while preserving its aspect ratio. Generally at least two edges of the text will meet the corresponding edges of the box.* **auto** — *This method tries to fit the text into the box automatically. In detail: Same as* `nofit` *if the text fits into the box. Otherwise a scaling factor is calculated such that the text fits into the box. If this factor is larger than the* `shrinklimit` *option the text is distorted to fit into the box, otherwise the* `meet` *method is applied.* **slice** — *Position the text according to the* `position` *option, and scale it such that it entirely covers the box, while preserving the aspect ratio and making sure that at least one dimension of the text is fully contained in the box. Generally parts of the text's other dimension will extend beyond the box, and will therefore be clipped.* **entire** — *Position the text according to the* `position` *option, and scale it such that it entirely covers the box. Generally this method will distort the text. The* `scale` *option will be ignored.*
font	*(Font handle) A font handle returned by PDF_load_font( ). Default: the current font*

Table 4.1 Options for PDF_fit_textline( ) and PDF_info_textline( )

key	explanation
fontname	*(Name string; must be used with the* `encoding` *option; will be ignored if* `font` *is specified) Name of font*
fontsize	*(Float, percentage or option list; required if the* `font` *option is provided) Size of the font, measured in units of the current user coordinate system. In PDF_fit_textline( ) percentages relate to the box width (for* `orientate=north` *and* `south`*) or height (for* `orientate=east` *and* `west`*). Textflow: percentages relate to the size of the preceding text. Default: the current font size.* *If an option list is provided it must contain a keyword and a number. The keyword describes the desired font metric, and the number contains the desired size value:* **ascender** *The number will be interpreted as ascender height.* **bodyheight** *The number will be interpreted as minimum distance between baselines, i.e. descenders and ascenders of adjacent lines may exactly touch if this value is used as leading. This is the default behavior if no keyword is provided.* **capheight** *The number will be interpreted as capital letter height.* **xheight** *The number will be interpreted as lowercase letter height.*
glyphcheck	*(Keyword) Specifies the glyph checking policy: what happens if text contains codes which cannot be mapped to a glyph in the selected font. Default*[1]*:* `replace`*.* **none** *No checking* **error** *An exception will be thrown for unavailable glyphs. A detailed error message can be retrieved with PDF_get_errmsg( ).* **replace** *PDFlib will try to replace unavailable glyphs with appropriate replacement glyphs; ligatures will be decomposed. If a suitable replacement is not available, the glyph will be replaced with* `replacementchar`*.*
glyphwarning	*Deprecated, use* `errorpolicy`
horizscaling	*(Float or percentage; must be different from 0) The horizontal text scaling (see Table 3.7). Default*[1] *for PDF_add/create_textflow( ): 100*
italicangle	*(Float) Specifies the italic (slant) angle of text in degrees. Default*[1] *for PDF_add/create_textflow( ): 0*
kerning	*(Boolean) Kerning behavior (see Table 3.7). Default: the global* `kerning` *parameter*

Table 4.1 Options for PDF_fit_textline( ) and PDF_info_textline( )

key	explanation
leader	*(Option list; will be ignored if* boxsize *is not specified) Specifies filler text (e.g. dot leaders) which be inserted repeatedly between the border of the text box and the text (default: no leader):*
	alignment *(One or two keywords) The first keyword specifies the alignment of the leader between the left border of the fitbox and the textline; the second keyword specifies the alignment of the leader between the textline and the right border of the fitbox. If only one keyword is specified it will be used for the leader between the textline and the right border of the fitbox. Supported keywords (default:* {none grid}*):*
	center *The leader is justified between the textline and the border of the fitbox.*
	grid *PDFlib snaps the position of the leader text to the next multiple of one half of the width of the leader text to the left or right of the textline. This may result in a gap between the textline and the leader text which spans at most 50% of the width of the leader text.*
	justify *The leader is justified between the textline and the border of the fitbox by applying suitable* charspacing*.*
	left *The leader starts at the left border of the fitbox or at the end of the textline.*
	none *No leader*
	right *The leader stops at the beginning of the textline or the right border of the fitbox.*
	encoding *(String; must be used with* fontname*; will be ignored if* font *is specified) Encoding name*
	fillcolor *(Color) Color of the leader. Default: color of the text line*
	font *(Font handle) Handle for the font to be used for the leader. Default: font of the text line*
	fontname *(Name string; must be used with* encoding*; will be ignored if* font *is specified) Name of the font for the leader*
	fontsize *(Float or option list) Size of the leader. Default: font size of the textline*
	text *(Content string) The text which will be used for the leader. Default: U+002E '.' (period)*
	yposition *(Float or keyword) Specifies the vertical position of the leader relative to the baseline, or via one of the keywords* fontsize, ascender, xheight, baseline, descender*. Default:* baseline
	In addition, all options of PDF_load_font( ) can be supplied.
locallink	*Deprecated; use the matchbox feature to create links in the text (see Section 4.4, »Matchboxes«, page 81)*
margin	*(List of floats) One or two float values describing additional horizontal and vertical extensions of the text box. Default: 0*
matchbox	*(Option list) Option list with matchbox details according to Table 4.12*
orientate	*(Keyword) Specifies the desired orientation of the text when it is placed. Default:* north*.*
	north *upright*
	east *pointing to the right*
	south *upside down*
	west *pointing to the left*
overline	*(Boolean) Overline mode (see Table 3.7). Default[1] for PDF_add/create_textflow( ):* false
position	*(List of floats or keywords) Alignment control: one or two values specifying the position of the reference point (x, y) within the text's bounding box, with {0 0} being the lower left corner of the text box, and {100 100} the upper right corner. If the* boxsize *option has been specified, the* position *option also specifies the positioning of the target box. The values are expressed as percentages of the text width and height. If both percentages are equal it is sufficient to specify a single float value.*
	The keywords left, center, right *(in x direction) or* bottom, center, top *(in y direction) can be used as equivalents for the values 0, 50, and 100. If only one keyword has been specified, the corresponding keyword for the other direction will be added. Default:* {left bottom}*. Examples:*
	{0 50} *or* {left center} — *left-justified text*
	{50 50} *or* {center} — *results in centered text*
	{100 50} *or* {right center} — *results in right-justified text*

Table 4.1 Options for PDF_fit_textline( ) and PDF_info_textline( )

key	explanation
rotate	*(Float) Rotate the coordinate system, using the reference point as center and the specified value as rotation angle in degrees. This results in the box and the text being rotated. The rotation will be reset when the text has been placed. Default: 0*
showborder	*(Boolean) If* `true`*, the border of the fitbox will be stroked (using the current graphics state). This may be useful for development and debugging. Default:* `false`
shrinklimit	*(Float or percentage) The lower limit of the shrinkage factor which will be applied to fit text with* `fitmethod=auto`*. Default: 0.75*
stamp	*(Keyword; will be ignored if* `boxsize` *is not specified) This option can be used to create a diagonal stamp within the box specified in the* `boxsize` *option. The text comprising the stamp will be as large as possible. The options* `position`*,* `fitmethod`*, and* `orientate` *(only* `north` *and* `south`*) will be honored when placing the stamp text in the box. Default:* none. **ll2ur** *The stamp will run diagonally from the lower left corner to the upper right corner.* **ul2lr** *The stamp will run diagonally from the upper left corner to the lower right corner.* **none** *No stamp will be created.*
strikeout	*(Boolean) Strikeout mode (see Table 3.7). Default[1] for PDF_add/create_textflow( ):* `false`
strokecolor	*(Color) Stroke color of the text. Default[1] for PDF_add/create_textflow( ):* `{gray 0}`
strokewidth	*(Float, percentage, or keyword) Line width for stroked text (absolute value or relative to the* `fontsize`*). The keyword* `auto` *specifies the value of* `underlinewidth` *for the font. Default:* `auto`
textformat	*(Keyword; only for non Unicode compatible language bindings) Format used to interpret the supplied text. Default: the global* `textformat` *parameter.*
textrendering	*(Integer) Text rendering mode (see Table 3.7). Default[1] for PDF_add/create_textflow( ): 0*
textrise	*(Float or percentage) Text rise mode (see Table 3.7). Percentages are based on* `fontsize`*. Default[1] for PDF_add/create_textflow( ): 0*
underline	*(Boolean) Underline mode (see Table 3.7). Default[1] for PDF_add/create_textflow( ):* `false`
underline-position	*(Float, percentage, or keyword) Position of the stroked line for underlined text relative to the baseline (absolute values or relative to the fontsize; a typical value is -10%). The keyword* `auto` *specifies a font-specific value which will be retrieved from the font metrics or outline file. Default:* `auto`
underline-width	*(Float, percentage, or keyword) Line width for underlined text (absolute value or relative to the fontsize; a typical value is 5%). The keyword* `auto` *or the value 0 specifies a font-specific value which will be retrieved from the font metrics or outline file. Default:* `auto`
weblink	*Deprecated; use the* `matchbox` *option in PDF_fit_textline( ) and PDF_create_annotation( ) to create links in the text (see Section 4.4, »Matchboxes«, page 81).*
wordspacing	*(Float or percentage) Word spacing (see Table 3.7). Percentages are based on fontsize. Default[1] for PDF_add/create_textflow( ): 0*
xadvancelist	*(List of floats) Specifies the advance width of all glyphs in the text in user coordinates. The length of the list must be less or equal than the number of glyphs in the text. The* `xadvance` *values will be used instead of the standard glyph widths. Other effects, such as kerning and character spacing, are unaffected.*

1. Default for PDF_fit_textline( ): the corresponding parameter in the current graphics state

C++ Java ***double info_textline(String text, String keyword, String optlist)***

Perl PHP ***float PDF_info_textline(resource p, string text, string keyword, string optlist)***

C ***double PDF_info_textline(PDF *p, const char *text, int len, const char *keyword, const char *optlist)***

Perform textline formatting and query the resulting metrics.

text (Content string) The contents of the textline.

len (C language binding only) The length of text in bytes, or 0 for null-terminated strings.

keyword A keyword specifying the requested information according to Table 4.2.

optlist An option list specifying textline options according to Table 4.1. Options which are not relevant for calculating the size of the text will silently be ignored.

Returns The value of some text metric value as requested by *keyword*.

Details This function will perform all calculations required for placing the text according to the supplied options, but will not actually create any output on the page. The text reference position is assumed to be {0 0}.

Scope any except *object*

Table 4.2 Keywords for PDF_info_textline( )

keyword	***explanation***
angle	*Rotation angle of the baseline in degree, i.e. the text rotation*
ascender ***capheight*** ***descender***	*Corresponding typographic value in user coordinates*
endx, endy	*x/y coordinates of the text end position in the user coordinate system*
height	*Height of the text string according to the* `boxheight` *specification of the matchbox*
perpendiculardir	*Unit vector perpendicular to writingdir; for standard horizontal text this would be (0, 1), for vertical text (1, 0)*
scalex, scaley	*Horizontal and vertical scaling factors. If these are different from 1 the text had to be scaled to fit into the box.*
startx, starty	*x/y coordinates of the text start position in the user coordinate system*
unmappedglyphs	*Number of glyphs in the text which could not be mapped*
width	*Width of the text string (in horizontal writing mode) or width of the widest glyph (in vertical writing mode)*
writingdirx ***writingdiry***	*x/y coordinates of the writing direction (direction of inline text progression), i.e. unit vector from (startx, starty) to (endx, endy). For standard horizontal text this would be (1, 0), for vertical text (0, -1)*
xheight	*Corresponding typographic value in user coordinates*

4.2 Multi-Line Text with Textflows

C++ Java ***int add_textflow(int textflow, String text, String optlist)***

Perl PHP ***int PDF_add_textflow(resource p, int textflow, string text, string optlist)***

C ***int PDF_add_textflow(PDF *p, int textflow, const char *text, int len, const char *optlist)***

Create a Textflow object, or add text and explicit options to an existing Textflow.

textflow Textflow handle returned by an earlier call to *PDF_create_textflow( )* or *PDF_add_textflow( )*, or -1 to create a new Textflow.

text (Content string) The contents of the Textflow. The text may not contain any inline options.

len (C language binding only) The length of text in bytes, or 0 for null-terminated strings.

optlist An option list specifying Textflow options according to Table 4.3. The following options can be used:

- General: *errorpolicy*
- Font-related options: *encoding, font, fontname, fontsize*
  Both of the options *fontname* and *encoding* (corresponding to the same-named parameters of *PDF_load_font( )*) can be used to select a font. Alternatively, the *font* option can be used to supply a font handle which has been created with an earlier call to *PDF_load_font( )*. If *font* is specified, the *fontname* and *encoding* options will be ignored. The *fontsize* option is required. If *fontsize* is specified as a percentage, it will be interpreted as a percentage of the previous fontsize; the initial value is 1.
- All options for *PDF_load_font( )* (see Table 3.4). These options will only be used if both the *fontname* and *encoding* options are supplied (but not with the *font* option: *ascender, autocidfont, autosubsetting, capheight, descender, embedding, fontstyle, keepnative, kerning, linegap, metadata, monospace, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*
- The following appearance options for *PDF_fit_textline( )* (see Table 4.1): *charref, charspacing, dasharray, escapesequence, fakebold, fillcolor, font, fontsize, glyphcheck, horizscaling, italicangle, kerning, matchbox, overline, strikeout, strokecolor, strokewidth, textformat, textrendering, textrise, underline, underlineposition, underlinewidth, wordspacing*
- Text semantics: *charclass, charmapping, hyphenchar, tabalignchar*
- Text formatting: *alignment, avoidemptybegin, fixedleading, hortabsize, hortabmethod, lastalignment, leader, leading, leftindent, minlinecount, parindent, rightindent, ruler, tabalignment*
- Options for controlling the line breaking algorithm: *adjustmethod, avoidbreak, maxspacing, minspacing, nofitlimit, shrinklimit, spreadlimit*
- Options which work as commands: *comment, mark, nextline, nextparagraph, resetfont, return, space*

Returns A Textflow handle which can be used in calls to *PDF_add_textflow( ), PDF_fit_textflow( ), PDF_info_textflow( )*, and *PDF_delete_textflow( )*. The handle is valid until the end of the enclosing *document* scope, or until *PDF_delete_textflow( )* is called with this handle.

If the *textflow* parameter is -1, a new Textflow will be created and the corresponding handle will be returned. Otherwise the handle supplied in the *textflow* parameter will be returned. By default, this function returns -1 (in PHP: 0) in case of an error. However, this behavior can be changed with the *errorpolicy* parameter or option. In case of an error the handle supplied in the *textflow* parameter can no longer be used in subsequent function calls (except in *PDF_delete_textflow( )* if it was different from -1).

Details This function processes the supplied text and creates an internal data structure from it. It determines text portions (e.g. words) which will later be used by the formatter, processes inline option lists, converts the text to Unicode if possible, determines potential line breaks, and calculates the width of text portions based on font and text options.

As opposed to *PDF_create_textflow( ),* which expects all text contents and options in a single call, this function is useful for supplying the text contents and options for a Textflow in separate calls. It will add the supplied *text* and *optlist* to a new or existing Textflow. Options specified in *optlist* will be evaluated before processing *text*. Both *text* and *optlist* may be empty.

If *textflow=-1* this function is almost equivalent to *PDF_create_textflow( )*. However, unlike *PDF_create_textflow( )* this function will not search for inline options in *text*. It is therefore not necessary to redefine the start character for inline option lists or to specify the length of the text with an inline option (not even for non-Unicode text and UTF-16 text).

This function does not create any output in the generated PDF document, but only prepares the text. Use *PDF_fit_textflow( )* to create output with the preprocessed Textflow handle.

By default, a new line will be forced by the characters U+000B (VT), U+2028 (LS), U+000A (LF), U+000D (CR), CRLF, U+0085 (NEL), U+2029 (PS), and U+000C (FF) in Unicode-compatible fonts. All of these except VT and LS force a new paragraph (which means that the *parindent* option will be effective). FF immediately stops the process of fitting text to the current fitbox (the function *PDF_fit_textflow( )* will be exited with a return string of *_nextpage).*

A horizontal tab character (HT) sets a new start position for subsequent text. The details of this are controlled by the *hortabmethod* and *hortabsize* options.

Soft hyphen characters (SHY) will be replaced with the character specified in the *hyphenchar* option if there is a line break after the soft hyphen.

Vertical writing mode is not supported.

Scope any except *object*

Table 4.3 Options for PDF_create_textflow( ), PDF_add_textflow( ), and inline options in PDF_create_textflow( )

option	explanation
adjustmethod	*(Keyword) Method used to adjust a line when a text portion doesn't fit into a line after compressing or expanding the distance between words subject to the limits specified by the* minspacing *and* maxspacing *options. Default:* auto.
	auto *The following methods are applied in order:* shrink, spread, nofit, split.
	clip *Same as* nofit, *except that the long part at the right edge of the fitbox (taking into account the* rightindent *option) will be clipped.*
	nofit *The last word will be moved to the next line provided the remaining (short) line will not be shorter than the percentage specified in the* nofitlimit *option. Even justified paragraphs may look slightly ragged.*
	shrink *If a word doesn't fit in the line the text will be compressed subject to* shrinklimit. *If it still doesn't fit the* nofit *method will be applied.*
	split *The last word will not be moved to the next line, but will forcefully be split after the last character in the box. For text fonts a hyphen character will be inserted, but not for symbol fonts or if* hyphenchar=none.
	spread *The last word will be moved to the next line and the remaining (short) line will be justified by increasing the distance between characters in a word, subject to* spreadlimit. *If justification still cannot be achieved the* nofit *method will be applied.*
alignment	*(Keyword) Specifies formatting for lines in a paragraph. Default:* left.
	left *Left-aligned, starting at* leftindent+parindent *(for the first line of a paragraph) and at* leftindent *(all other lines)*
	center *Centered between* leftindent *and* rightindent
	right *Right-aligned, ending at* rightindent
	justify *Left- and right-aligned*
avoidbreak[1]	*(Boolean) If* true, *try to avoid any line breaks until* avoidbreak *is reset to* false**.** *In other words, the text will be treated as in a non-Unicode compatible font. The splitting procedure is not affected by this option. Default:* false
avoid-emptybegin	*(Boolean) If* true, *empty lines at the beginning of a fitbox will be deleted. Default:* false
charclass[1]	*(List of pairs, where the first element in each pair is a keyword, and the second element is either a unichar or a list of unichars) The specified unichars will be classified by the specified keyword to determine the line breaking behaviour of those character(s):*
	letter *behave like a letter (e.g. a B)*
	punct *behave like a punctuation character (e.g. + / ; :)*
	open *behave like an open parenthesis (e.g. [ )*
	close *behave like a close parenthesis (e.g.])*
	default *reset all character classes to PDFlib's builtin defaults*
	Example: charclass={ close » open « letter={/ : =} punct & }

Table 4.3 Options for PDF_create_textflow( ), PDF_add_textflow( ), and inline options in PDF_create_textflow( )

option	explanation

charmapping[1]	*(List of pairs, where each pair either contains two unichars or a unichar and a list of unichar and integer) Replace individual characters with one or more instances of another character. The option list contains one or more pairs of Unichars. The first character in each pair will be replaced with the second character. Instead of one-to-one mapping the second element in each pair may be an option list containing a unichar and a count:*
	count > 0 *The replacement character will be repeated count times.*
	count < 0 *A sequence of multiple instances of the character will be reduced to the absolute value of the specified number.*
	count = 0 *The character will be deleted.*
	Examples:
	`charmapping={ hortab space  CRLF space  LF space  CR space }`
	`charmapping={ shy {shy 0} }`
	`charmapping={ hortab {space 4} }`

comment	*(String) Arbitrary text which will be ignored; useful for commenting option lists or macros*
encoding	*(String; must be used with the* `fontname` *option; will be ignored if* `font` *is specified) Encoding name*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
fixedleading	*(Boolean) If* `true`*, the first leading value found in each line will be used. Otherwise the maximum of all leading values in the line will be used. Default:* `false`
fontname	*(Name string; must be used with the* `encoding` *option; will be ignored if* `font` *is specified) Name of the font; remember to put* { *and* } *around font names which contain space characters.*
hortabsize[1]	*(Float or percentage) Width of a horizontal tab*[2]*. The interpretation depends on the* `hortabmethod` *option. Default: 7.5%*
hortabmethod[1]	*(Keyword) Treatment of horizontal tabs in the text. If the calculated position is to the left of the current text position, the tab will be ignored. Default:* `relative`*.* **relative** *The position will be advanced by the amount specified in* `hortabsize`*.* **typewriter** *The position will be advanced to the next multiple of* `hortabsize`*.* **ruler** *The position will be advanced to the n-th tab value in the* `ruler` *option, where n is the number of tabs found in the line so far. If n is larger than the number of tab positions the relative method will be applied.*
hyphenchar[1]	*(Unichar or keyword) Character which replaces a soft hyphen at line breaks. The value 0 and the keyword* none *completely suppress hyphens. Default: U+00AD (soft hyphen) if available in the font, U+002D (hyphen-minus) otherwise*
lastalignment	*(Keyword) Formatting for the last line in a paragraph. All keywords of the* `alignment` *option are supported, plus the following (default:* `auto`*):* **auto** *Use the value of the* `alignment` *option unless it is justify. In the latter case* `left` *will be used.*

Table 4.3 Options for PDF_create_textflow( ), PDF_add_textflow( ), and inline options in PDF_create_textflow( )

option	explanation
leader	*(Option list) Specifies filler text (e.g. dot leaders) which will be inserted repeatedly. Leaders will be inserted until the next tab position, or the end of the line if no tab is available. Leaders never span more than one line (default: no leader):* ***alignment*** *(Keyword) Alignment of the leader (default:* `grid`*):* ***center*** *The leader is centered between the last text fragment (or the start of the line if there is no text) and the tab position (or the end of the line if there is no tab).* ***grid*** *PDFlib will snap the position of the first leader character to the next multiple of one half of the width of the leader text. This may result in a gap between the text and the leader of at most 50% of the width of the leader.* ***justify*** *The leader is justified between the last text fragment (or the start of the line if there is no text) and the tab position (or the end of the line if there is no tab) by applying suitable charspacing.* ***left*** *The leader starts immediately after the last text fragment (or the start of the line if there is no text).* ***right*** *The leader stops immediately before the tab position (or the end of the line if there is no tab).* *The following suboptions of the* `leader` *option in PDF_fit_textline( ) are also supported (see Table 4.1):* ***encoding***, ***fillcolor***, ***font***, ***fontname***, ***fontsize***, ***text***, ***yposition***.
leading	*(Float or percentage) Distance between adjacent text baselines[3]. The actual value will be determined as follows: if there are option lists at the beginning of a line, the leading will be determined by the last relevant option* (`font`, `fontsize`, `leading`, *etc.). If there are additional option lists on the same line, any options relevant for leading will only be taken into account if* `fixedleading=false`*. If there are no option lists in the line at all, the previous leading value will be taken into account. Default: 100%*
leftindent	*(Float or percentage) Left indent of text lines[2]. If* `leftindent` *is specified within a line and the resulting position is to the left of the current text position, this option will be ignored for this line. Default: 0*
mark	*(Integer) Store the supplied number internally as a mark. The mark which has been stored least recently can later be retrieved with PDF_info_textflow( ). This may be useful for determining which portions of text have already been placed on the page.*
matchbox	*(Option list) Option list with matchbox details according to Table 4.12*
maxspacing[1] ***minspacing***[1]	*(Float or percentage) Maximum or minimum distance between words (in user coordinates, or as a percentage of the width of the space character). The calculated word spacing is limited by the provided values (but the* `wordspacing` *option will still be added). Defaults:* `minspacing=50%`*,* `maxspacing=500%`
minlinecount	*(Integer) Minimum number of lines in the last paragraph in the fitbox. If there are fewer lines they will be placed in the next fitbox. The value 2 can be used to prevent single lines of a paragraph at the end of a fitbox (»orphans«). Default: 1*
nextline ***nextparagraph***	*(Boolean) Force a new line or paragraph, even in fonts which are not Unicode-compatible.*
nofitlimit	*(Float or percentage) Lower limit for the length of a line with the* `nofit` *method[2]. Default: 75%.*
parindent	*(Float or percentage) Left indent of the first line of a paragraph[2]. The amount will be added to* `leftindent`*. Specifying this option within a line will act like a tab. Default: 0*
resetfont	*(Boolean) Reset* `font` *and* `fontsize` *to their previous values. This may be useful to reset the font after inserts, such as italic text. The* `font` *option has precedence over this option. This command only makes sense after the second setting of any font-related parameters, and will be ignored otherwise.*
return	*(String; must not start with an underscore _ character) Exit PDF_create_textflow( ) or PDF_add_textflow( ) with the supplied string as return value.*
rightindent	*(Float or percentage) Right indent of all text lines[2]. Default: 0*

Table 4.3 Options for PDF_create_textflow( ), PDF_add_textflow( ), and inline options in PDF_create_textflow( )

option	explanation
ruler[1]	*(List of floats or percentages) List of absolute tab positions for* `hortabmethod=ruler`[2]*. The list may contain up to 32 non-negative entries in ascending order. Default: integer multiples of* `hortabsize`
shrinklimit	*(Percentage) Lower limit for compressing text with* `adjustmethod=shrink`*; the calculated shrinking factor is limited by the provided value, but will be multiplied with the* `horizscaling` *option. Default: 85%*
space	*(Float or percentage) The text position will be advanced by the provided value*[3]*. This also works in fonts which are not Unicode-compatible.*
spreadlimit[1]	*(Float or percentage) Upper limit for the distance between characters for the spread method*[3]*; the calculated distance will be added to the value of the* `charspacing` *option. Default: 0*
tabalignchar[1]	*(Unichar) Character at which decimal tabs will be aligned. Default: U+002E '.'*
tabalignment[1]	*(List of keywords) Alignment for tab stops. Each entry in the list defines the alignment for the corresponding entry in the* `ruler` *option. Default:* `left`. **center** *Text will be centered at the tab position.* **decimal** *The first instance of* `tabalignchar` *will be left-aligned at the tab position. If no* `tabalignchar` *is found, right alignment will be used instead.* **left** *Text will be left-aligned at the tab position.* **right** *Text will be right-aligned at the tab position.*
textwarning	*Deprecated, use* `errorpolicy`

1. This option does not affect text in fonts which are not Unicode-compatible.
2. In user coordinates, or as a percentage of the width of the fitbox
3. In user coordinates, or as a percentage of the font size

Macros for Textflow options. Option lists for Textflows (either in the *optlist* parameter of *PDF_create_textflow( )* or *PDF_add_textflow( ),* or inline in the text supplied to *PDF_create_textflow( ))* may contain macro definitions and macro calls according to Table 4.4. Macros may be useful for having a central definition of multiply used option values, such as font names, indentation amounts, etc. Before parsing an option list each contained macros will be substituted with the contents of the corresponding option list provided in the macro definition. The resulting option list will then be parsed. The following example demonstrates a macro definition for two macros:

```
<macro {
        comment { The following macros are used as paragraph styles }
        H1 {fontname=Helvetica-Bold encoding=winansi fontsize=14 }
        body {fontname=Helvetica encoding=winansi fontsize=12 }
}>
```

These macros could be used as follows in an option list:

```
<&H1>Chapter 1
<&body>This chapter talks about...
```

The following rules apply to macro definition and use:

- Macros may be nested to an arbitrary depth (macro definitions may contain calls to other macros).
- Macros can not be used in the same option list where they are defined. In *PDF_create_textflow( )* a new inline option list which uses the macro can be started immediately after the end of the inline option list in which the macro is defined. When using *PDF_*

add_textflow( ) one function call is required to define the macro, and another one to use it (since *PDF_add_textflow( )* accepts only a single option list at a time).
- Macro names are case-insensitive.
- Undefined macros will result in an exception.
- Macros can be redefined at any time.

Table 4.4 Option list macro definitions and calls for PDF_add_textflow( ), PDF_create_textflow( ), and fit_textflow( )

option	explanation
comment	*(String) Arbitrary text which will be ignored; useful for commenting macros*
macro	*(List of pairs) Each pair describes the name and definition of a macro as follows:* **name** *(string) The name of the macro which can later be used for macro calls. Macros which have already been defined can be redefined later. The special name comment will be ignored.* **suboptlist** *An option list which will literally replace the macro name when the macro is called. Leading and trailing whitespace will be ignored.*
&name	*The macro with the specified name will be expanded, and the macro name (including the & character) will be replaced by the macro's contents, i.e. the suboptlist which has been defined for the macro (without the surrounding braces). The macro name is terminated by whitespace, {, }, =, or &. Therefore, these characters can not be used as part of a macro name.* *Nested macros will be expanded without any nesting limit. Macros contained in string options will also be expanded. Macro substitution must result in a valid option list.*

C++ Java ***int create_textflow(String text, String optlist)***
Perl PHP ***int PDF_create_textflow(resource p, string text, string optlist)***
C ***int PDF_create_textflow(PDF *p, const char *text, int len, const char *optlist)***

Create a Textflow object from text contents, inline options, and explicit options.

text (Content string) The contents of the Textflow. It may contain text in various encodings, macros (see »Macros for Textflow options«, page 65), and inline option lists according to Table 4.3 and Table 4.5 (see also »Inline option lists for Textflows«, page 67). If *text* is an empty string, a valid Textflow handle will be returned nevertheless.

len (C language binding only) The length of text in bytes, or 0 for null-terminated strings.

optlist An option list specifying Textflow options. Options specified in the *optlist* parameter will be evaluated before those in inline option lists in *text* so that inline options have precedence over options provided in the *optlist* parameter. The following options can be used:
- All options of *PDF_add_textflow( )* (see option list of *PDF_add_textflow( )* and Table 4.3)
- Options for controlling inline option list processing according to Table 4.5: *begoptlistchar, endoptlistchar, fixedtextformat, textlen*

Returns A Textflow handle which can be used in calls to *PDF_add_textflow( ), PDF_fit_textflow( ), PDF_info_textflow( ),* and *PDF_delete_textflow( ).* The handle is valid until the end of the enclosing *document* scope, or until *PDF_delete_textflow( )* is called with this handle.By default this function returns -1 (in PHP: 0) in case of an error. However, this behavior can be changed with the *errorpolicy* parameter or option.

Details This function accepts options and text to be prepared for Textflow. Unlike *PDF_add_textflow( )* the text may contain inline options. Searching for inline option lists can be disabled for parts or all of the text by supplying the *textlen* option in the *optlist* parameter (see »Inline option lists for Textflows«, page 67).

This function does not create any output in the generated PDF document, but only prepares the text according to the supplied options. Use *PDF_fit_textflow( )* to create output with the resulting Textflow handle.

See the *Details* section of *PDF_add_textflow( )* for more information regarding special characters, line breaking, etc.

Scope any except *object*

Table 4.5 Additional options for inline option list processing in PDF_create_textflow( )

option	*explanation*
begoptlistchar	*(Unichar or keyword) Character which starts inline option lists. Replacing the default character may be useful if this character appears in the text literally (see »Inline option lists for Textflows«, page 67). If* `textlen` *is not specified, the* `begoptlistchar` *character in the text must be encoded in the same text format and encoding as the preceding text. This means that the Unicode value of* `begoptlistchar` *must be chosen such that it is contained in the encoding of the preceding text. The keyword* `none` *can be used to completely disable the search for option lists. Default: U+003C (<)*
endoptlistchar	*(Unichar; U+007D '}' is not allowed) Character which terminates inline option lists. Default: U+003F (>)*
fixedtextformat	*(Boolean; will be ignored in Unicode-aware language bindings; this option doesn't make sense in inline option lists, and can only be used in the* `optlist` *parameter) If* `true`*, all text fragments and inline options lists will use the same* `textformat`*, which must be one of* `utf8`, `utf16`, `utf16be`*, or* `utf16le`*. This is useful if text and inline options come from the same source.* *If* `false`*, inline option lists including the delimiters must be encoded in* `textformat=bytes`*, regardless of the format used for the actual text. This allows the combination e.g. of UTF-16 text with ASCII-encoded inline option lists (the text may come from a Unicode database, while inline options are constructed as ASCII text within the application). Default:* `false`
textlen	*(Integer or keyword; required for text in fonts which are not Unicode-compatible, or for text fragments with* `fixedtextformat=false` *and* `textformat=utf16xx` *in non-Unicode aware languages) Number of bytes or (in Unicode-aware languages) characters before the next inline option list (see »Inline option lists for Textflows«, page 67). The characters are counted before character references are resolved, e.g.* `<textlen=8>&#x2460;<...>`*. The keyword* `all` *specifies all of the remaining text. Default: the text will be searched for the next occurrence of* `begoptlistchar`*.*

Inline option lists for Textflows. The content provided in the *text* parameter of *PDF_create_textflow( )* (but not *PDF_add_textflow( ))* may include an arbitrary number of option lists (inline options) specifying Textflow options according to Table 4.3. All of these options can alternatively be provided in the *optlist* parameter of *PDF_create_textflow( )* and *PDF_add_textflow( )*. The same option can be specified multiply in a single option list; in this case only the last occurrence of an option will be taken into account.

Inline option lists must be enclosed with the characters specified in the *begoptlistchar* and *endoptlistchar* options (by default: < and >). Obviously, conflicts could arise if the character used for starting inline option lists must also be used in the actual text. There are several methods to resolve this conflict, depending on whether or not the text contains any inline option lists. Remember that *PDF_add_textflow( )* completely separates text and options, so the conflict doesn't arise there.

If the text does not contain any inline options lists you can completely disable the search for inline option lists by one of the following methods:

- Set *begoptlistchar=none* in the *optlist* parameter of *PDF_create_textflow( )*.
- Set the *textlen* option in the *optlist* parameter of *PDF_create_textflow( )* to the length of the full text.

If the text actually contains inline option lists you can avoid the conflict between text contents and the *begoptlistchar* for starting an inline option list by using one of the following methods:

- Replace all occurrences of the < character in the text with the corresponding numeric or character entity reference (*<* or *<)* and start inline option lists with the literal < character:

```
A&lt;B<fontname=Helvetica encoding=winansi>
```

- Set the *begoptlistchar* option in the *optlist* parameter of *PDF_create_textflow( )* or an inline option list to a character which is not used in the text (e.g. *$)*, and use this character to start inline option lists:

```
<begoptlistchar=$>A<B$fontname=Helvetica encoding=winansi>
```

- Specify the length of the next text fragment (until the start of the next inline option list) in the preceding inline option list using the *textlen* option:

```
<textlen=3>A<B<fontname=Helvetica encoding=winansi>
```

Note *If an inline option list is provided immediately after another option list, they are assumed to enclose a text fragment of zero length. This is important when supplying the* `textlen` *option in the first option list.*

C++ Java ***String fit_textflow(int textflow, double llx, double lly, double urx, double ury, String optlist)***

Perl PHP ***string PDF_fit_textflow(resource p, int textflow, float llx, float lly, float urx, float ury, string optlist)***

C ***const char *PDF_fit_textflow(PDF *p,***
int textflow, double llx, double lly, double urx, double ury, const char *optlist)

Format the next portion of a Textflow into a rectangular area.

textflow A Textflow handle returned by a call to *PDF_create_textflow( )* or *PDF_add_textflow( )*.

llx, lly, urx, ury *x* and *y* coordinates of the lower left and upper right corners of the target rectangle (the *fitbox)* in user coordinates. The corners can also be specified in reverse order.

optlist An option list specifying processing options according to Table 4.6. The following options can be used:
blind, featherlimit, firstlinedist, fitmethod, fontscale, keep, lastlinedist, linespreadlimit, maxlines, minfontsize, orientate, rewind, rotate, showborder, showtabs, verticalalign, wrap

Returns A string which specifies the reason for returning from the function:

- *_stop:* all text in the Textflow has been processed.
- *_nextpage:* Waiting for the next page (caused by a form feed character U+000C). Another call to *PDF_fit_textflow( )* is required for processing the remaining text.
- *_boxfull:* No more space is available in the fitbox, or the maximum number of lines (as specified via the *maxlines* option) has been placed in the box, or *fitmethod=auto*

and *minfontsize* has been specified but the text didn't fit into the box. Another call to *PDF_fit_textflow( )* is required for processing the remaining text.

- *_boxempty:* The box doesn't contain any text at all after processing. This may happen if the size of the box is too small to hold any text. No more calls to *PDF_fit_textflow( )* with the same fitbox should be issued in order to avoid infinite loops.
- Any other string: The string supplied to the *return* command in an inline option list.

If there are multiple simultaneous reasons for returning, the first in the list (from top to bottom) will be reported. The returned string is valid until the next call to this function.

Details The current text and graphics states do not influence the text output created by this function (this is different from *PDF_fit_textline( )*). Use *fillcolor*, *strokecolor* and other appearance options (see Table 4.1) in *PDF_create_textflow( )* or *PDF_add_textflow( )* to control the appearance of the text. After returning from this function the text state will be unchanged. However, the *textx/texty* parameters will be adjusted to point to the end of the generated text output (unless the *blind* option has been set to *true)*.

Scope *page, pattern, template, glyph*

Table 4.6 Options for PDF_fit_textflow( )

option	*explanation*
blind	*(Boolean) If* `true`*, no output will be generated, but all calculations will be performed and the formatting results can be checked with PDF_info_textflow( ). Default:* `false`
firstlinedist	*(Float, percentage, or keyword) Distance between the top of the fitbox and the baseline for the first line of text, specified in user coordinates, as a percentage of the relevant font size (the first font size in the line if* `fixedleading=true`*, and the maximum of all font sizes in the line otherwise), or as a keyword. Default:* `leading`*.* ***leading*** *The leading value determined for the first line; typical diacritical characters such as À will touch the top of the fitbox.* ***ascender*** *The ascender value determined for the first line; typical characters with larger ascenders, such as d and h will touch the top of the fitbox.* ***capheight*** *The capheight value determined for the first line; typical capital uppercase characters such as H will touch the top of the fitbox.* ***xheight*** *The xheight value determined for the first line; typical lowercase characters such as x will touch the top of the fitbox.* *If* `fixedleading=false` *the maximum of all* `leading`*,* `ascender`*,* `xheight`*, or* `capheight` *values found in the first line will be used.*
fitmethod	*(Keyword) Specifies the method used to fit the text into the fitbox. Default:* `clip` ***auto*** *PDF_fit_textflow( ) will repeatedly be called in blind mode with reduced font size and other font-related options (see* `fontscale`*) until the text fits into the fitbox (but see also option* `minfontsize`*)* ***clip*** *The text will be truncated at the bottom of the fitbox.* ***nofit*** *The text can extend beyond the bottom of the fitbox.*
fontscale	*(Float or percentage) Values of* `fontsize` *and absolute values (but not percentages) of* `leading`*,* `minspacing`*,* `maxspacing`*,* `spreadlimit`*, and* `space` *will be multiplied with the supplied scaling factor or percentage. Default: 1 if* `rewind=0`*, otherwise the value supplied with the corresponding call to PDF_fit_textflow( ).*

Table 4.6 Options for PDF_fit_textflow( )

option	explanation
lastlinedist	*(Float, percentage, or keyword; will be ignored for* `fitmethod=nofit`*) Minimum distance between the baseline for the last line of text and the bottom of the fitbox, specified in user coordinates, as a percentage of the font size (the first font size in the line if* `fixedleading=true`*, and the maximum of all font sizes in the line otherwise), or as a keyword. Default: 0, i.e. the bottom of the fitbox will be used as baseline, and typical descenders will extend below the fitbox. The following keyword can be used:* **descender** *The descender value determined for the last line; typical characters with descenders, such as g and j will touch the bottom of the fitbox.* *If* `fixedleading=false` *the maximum of all descender values found in the last line will be used.*
linespread-limit	*(Float or percentage; only for* `verticalalign=justify`*) Maximum amount in user coordinates or as percentage of the leading for increasing the leading for vertical justification. Default: 200%*
maxlines	*(Integer or keyword) Maximum number of lines in the fitbox, or the keyword* `auto` *which means that as many lines as possible will be placed in the fitbox. When the maximum number of lines has been placed PDF_fit_textflow( ) will return the string* `_boxfull`*. Default:* `auto`
minfontsize	*(Float or percentage) Minimum font size allowed when text is scaled down to fit into the box, especially for* `fitmethod=auto`*. The limit is specified in user coordinates or as a percentage of the height of the fitbox. If the limit is and the text still does not fit the string* `_boxfull` *will be returned. Default: 0.01%*
orientate	*(Keyword) Specifies the desired orientation of the text when it is placed. Default:* `north`*.* **north** *upright* **east** *pointing to the right* **south** *upside down* **west** *pointing to the left*
rewind	*(Integer: -2, -1, 0, or 1) State of the supplied Textflow is reset to the state before some other call to PDF_fit_textflow( ) with the same Textflow handle. Default: 0.* **1** *Rewind to the state before the first call to PDF_fit_textflow( ).* **0** *Don't reset the Textflow.* **-1** *Rewind to the state before the last call to PDF_fit_textflow( ).* **-2** *Rewind to the state before the second last call to PDF_fit_textflow( ).*
rotate	*(Float) Rotate the coordinate system, using the lower left corner of the fitbox as center and the specified value as rotation angle in degrees. This results in the box and the text being rotated. The rotation will be reset when the text has been placed. Default: 0*
showborder	*(Boolean) If* `true`*, the border of the fitbox will be stroked (using the current graphics state). This may be useful for development and debugging. Default:* `false`
showtabs	*(Keyword) Tab stops and left indents will be visualized with vertical lines as a debugging aid. The lines will be drawn according to the graphics state which was active before calling PDF_fit_textflow( ) (default:* `none`*):* **none** *no lines will be drawn* **fitbox** *lines will be drawn over the full height of the fitbox* **validarea** *lines will be drawn only in vertical area where they are valid*

Table 4.6 Options for PDF_fit_textflow( )

option	explanation
verticalalign	*(Keyword) Vertical alignment of the text in the fitbox; the* `firstlinedist` *and* `lastlinedist` *options will be taken into account as appropriate (default:* `top`*):*
	top *Formatting will start at the first line, and continue downwards. If the text doesn't fill the fitbox there may be whitespace below the text.*
	center *The text will be vertically centered in the fitbox. If the text doesn't fill the fitbox there may be whitespace both above and below the text.*
	bottom *Formatting will start at the last line, and continue upwards. If the text doesn't fill the fitbox there may be whitespace above the text.*
	justify *The text will be aligned with top and bottom of the fitbox. In order to achieve this the leading will be increased up to the limit specified by linespreadlimit. The height of the first line will only be increased if* `firstlinedist=leading`*.*
wrap	*(Option list) The text will run around (wrap) the union of the specified shapes, i.e. text will appear within the fitbox, but only outside the specified object shapes. This can be used to place graphics within the Textflow and wrap the text around it. Absolute coordinate values will be interpreted in the user coordinate system; percentages will be interpreted in the fitbox coordinate system, i.e. the lower left corner of the fitbox is (0, 0), the upper right corner is (100, 100). Up to 32 percentage values can be used. The specified shapes should not overlap; the behavior in overlapping areas is undefined. The following options are supported:*
	boxes *(List of rectangles) One or more rectangles.*
	usematchboxes *(List of string lists) The first element in each list is a name string which specifies a matchbox. The second element is either an integer specifying the number of the desired rectangle, or the keyword* `all` *to specify all rectangles referring to the selected matchbox. If the second element is missing, it defaults to* `all`*. The bounding box of each rectangle will be used as shape for text wrapping.*
	Examples:
	Exclude the upper right quarter of the fitbox: `wrap={ boxes={50% 50% 100% 100%} }`
	Exclude the area of an image with a matchbox called `im1:` `wrap={ usematchboxes={ im } }`

C++ Java double info_textflow(int textflow, String keyword)
Perl PHP float PDF_info_textflow(resource p, int textflow, string keyword)
C double PDF_info_textflow(PDF *p, int textflow, const char *keyword)

Query the current state of a Textflow.

textflow A Textflow handle returned by a call to *PDF_add/create_textflow( )* or *PDF_fill_textblock( )* with the *textflowhandle* option.

keyword A keyword specifying the requested information according to Table 4.7.

Returns The value of some Textflow parameter as requested by *keyword*. This function returns correct geometry information even in blind mode (unlike the *textx/texty* parameters).

Scope any except *object*

Table 4.7 Keywords for PDF_info_textflow( )

keyword	explanation
boxlinecount	*Number of lines in the last fitbox*
firstparalinecount	*Number of lines in the first paragraph of the fitbox*

Table 4.7 Keywords for PDF_info_textflow( )

keyword	explanation
firstlinedist	*Distance between the first text baseline and the fictitious baseline above (if* `fitmethod=top` *this will be the upper border of fitbox)*
lastmark	*Number of the last mark found in the processed part of the Textflow in the last fitbox (marks can be set with the* `mark` *option)*
lastlinedist	*Distance between the the last text baseline and the fictitious baseline below, assuming unmodified leading (if* `fitmethod=bottom` *this will be the lower border of the fitbox)*
leading	*The current value of the* `leading` *option, as determined by the text and options within the Textflow*
lastparalinecount	*Number of lines in the last paragraph of the fitbox*
leftlinex[1], leftliney[1]	*The x and y coordinates of the line with the leftmost start in the most recently filled fitbox, in current user coordinates*
maxlinelength	*Length of the longest text line in the most recently filled fitbox*
maxliney[1]	*The y coordinate of the baseline of the longest text line in the most recently filled fitbox, in current user coordinates*
minlinelength	*Length of the shortest text line in the most recently filled fitbox*
minliney[1]	*The y coordinate of the baseline of the shortest text line in the most recently filled fitbox, in current user coordinates*
remainchars	*(Deprecated) Number of characters not yet processed. This count does not include the number of characters in inline option lists and character references. The value may be unreliable because of various text substitution processes (e.g. CR/NL combinations).*
returnreason	*String index (see Table 2.1) for the return reason of the most recent direct or indirect call to PDF_fit_textflow( ). The string will be one of the return strings of PDF_fit_textflow( ). This is useful for querying the result of indirect Textflow calls issued internally by PDF_fill_textblock( ).*
rightlinex[1], rightliney[1]	*The x and y coordinates of the line with the rightmost end in the most recently filled fitbox, in current user coordinates*
split	*Specifies whether text splitting occurred in the last fixbox:* **0** *No text line had to be split.* **1** *At least one text line had to be split.*
textendx, textendy	*The x or y coordinate of the current text position after the most recently filled fitbox in current user coordinates*
textheight	*Height of the bounding box of the whole text (taking* `firstlinedist` *and* `lastlinedist` *into account) in current user coordinates*
textwidth	*Width of the bounding box of the whole text in current user coordinates*
used	*Percentage of text (0...100) which has been placed so far*
x1, y1, ... ,x4, y4	*Coordinates of the bounding box of the whole text (taking* `firstlinedist` *and* `lastlinedist` *into account) in current user coordinates*

1. If `rotate` *is different from 0 this value referes to the rotated system.*

C++ Java ***void delete_textflow(int textflow)***

Perl PHP ***PDF_delete_textflow(resource p, int textflow)***

C ***void PDF_delete_textflow(PDF *p, int textflow)***

Delete a Textflow and all associated data structures.

textflow A Textflow handle returned by a call to *PDF_create_textflow( )* or *PDF_add_textflow( )*.

Details Textflows which have not been deleted with this function will be deleted automatically at the end of the enclosing *document* scope. However, failing to call *PDF_delete_textflow( )* may significantly slow down the application if many Textflows are generated.

Scope any

4.3 Table Formatting

C++ Java ***int add_table_cell(int table, int column, int row, string text, string optlist)***

Perl PHP ***int PDF_add_table_cell(resource p, int table, int column, int row, string text, string optlist)***

C ***int PDF_add_table_cell(PDF *p,***
int table, int column, int row, const char *text, int len, const char *optlist)

Add a cell to a new or existing table.

table A valid table handle retrieved with another call to *PDF_add_table_cell( )*, or -1 (in PHP: 0) for the first call. The table handle must not yet have been used in a call to *PDF_fit_table( )*, i.e. all table contents must be defined before placing the table on the page.

column, row Number of the column and row containing the cell. If the cell spans multiple columns and/or rows the numbers of the leftmost column and the topmost row must be supplied. The first column/row has number 1.

text (Content string) Text for filling the cell. If *text* is not empty it will be used for filling the cell with *PDF_fit_textline( )*.

len (C language binding only) Length of *text* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

optlist An option list specifying table cell formatting details according to Table 4.8. The following options can be used:

- General: *errorpolicy*
- Column and row definition: *colwidth, colscalegroup, minrowheight, return, rowheight, rowjoingroup, rowscalegroup*
- Cell definition: *checkwordsplitting, colspan, margin, marginleft, marginbottom, marginright, margintop, matchbox, rowspan*
- Cell contents: *fittextline, textflow, fittextflow, image, fitimage, pdipage, fitpdipage*

Returns A table handle which can be used in subsequent table-related calls. The return value must be checked for -1 (in PHP: 0) which signals an error. In case of an error only the last cell definition will be discarded; no contents will be added to the table, but the table handle is still valid. The returned table handle can not be reused across multiple PDF output documents.

Details A table cell can be filled with images, imported PDF pages, Textflows, or textlines. Multiple content types can be specified for a particular cell.

Scope any except *object*

Table 4.8 Options for PDF_add_table_cell( )

key	explanation
checkword-splitting	*(Boolean) If* `true`, *the table formatting algorithm will check whether Textflow cells require at least one forced word splitting. If so, the cell width will be increased in an attempt to avoid word splittings. Default:* `true`
colscale-group[1]	*(String) Name of a column group to which the column will be added. All columns in a group will be scaled uniformly if one of the columns in the group must be enlarged to completely hold long text. If a cell spans multiple columns the affected columns form a scale group automatically.*

Table 4.8 Options for PDF_add_table_cell( )

key	explanation
colspan	*(Integer) Number of columns spanned by the cell. Default: 1*
colwidth[1]	*(Float or percentage) Width of the column specified in the* `column` *parameter, specified in user coordinates or as a percentage of the table's first fitbox (see PDF_fit_table( )). If percentages are used this option is required, and all column widths must be specified as percentages.*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
fitimage	*(Option list; only relevant for images) Option list for PDF_fit_image( ). This option list will be applied to place the supplied image in the cell. The lower left corner of the inner cell box will be used as the reference point.* *Default:* `boxsize={<width> <height>} fitmethod=meet position=center`*, where* `<width>` *and* `<height>` *are the calculated width and height of the inner cell box. This calculated option list will be prepended to the user-specified option list. The box size will be calculated automatically; any* `boxsize` *option in the supplied option list will be ignored.*
fitpdipage	*(Option list; only relevant for PDI pages) Option list for PDF_fit_pdi_page( ). This option list will applied to place the supplied page in the cell. The lower left corner of the inner cell box will be used as the reference point.* *Default:* `boxsize={<width> <height>} fitmethod=meet position=center`*, where* `<width>` *and* `<height>` *are the calculated width and height of the inner cell box. This calculated option list will be prepended to the user-specified option list. The box size will be calculated automatically and will be ignored in the supplied option list.*
fittextflow	*(Option list; only relevant for Textflows) Option list for PDF_fit_textflow( ). This option list will be applied to place the supplied Textflow in the cell. The inner cell box will be used as fitbox.* *Default:* `verticalalign=center lastlinedist=descender`*. This option list will be prepended to the user-specified option list.*
fittextline	*(Option list; only relevant for textlines) Option list for PDF_fit_textline( ). This option list will be applied to fit the supplied text into the cell. The lower left corner of the inner cell box will be used as the reference point. Options which have not been specified will be replaced with the respective defaults; the current text state is not taken into account.* *Default:* `boxsize={<width> <height>} fitmethod=nofit position=center`*, where* `<width>` *and* `<height>` *are the calculated width and height of the inner cell box. This calculated option list will be prepended to the supplied option list. The box size will be calculated automatically; any* `boxsize` *option in the supplied option list will be ignored.*
image	*(Image handle) The image associated with the handle will be placed in the inner cell box.*
margin **marginleft** **marginbottom** **marginright** **margintop**	*(Float or percentage) Left/bottom/right/top cell margins in user coordinates (must be greater than or equal to 0) or as a percentage of the cell width or height (must be less than 100%). The specified margins define the inner cell box which serves as the fitbox for the cell contents. Default for* `margin`*: 0; Default for all others:* `margin`
matchbox	*(Option list) Option list with matchbox details according to Table 4.12.*
minrowheight[1]	*(Float or percentage) If a row cannot completely be placed in a table instance, this option specifies whether the row can be split and how small the fragments can get. The minimum fragment size can be specified in user coordinates or as a percentage of the row height. Default: 100%, i.e. no splitting*
pdipage	*(Page handle) The imported PDF page associated with the handle will be placed in the inner cell box. Default: none*
return[1]	*(String) PDF_fit_table( ) will stop after placing the specified row, and will return the specified string. The string must not start with an underscore character '_'. If the specified row is part of a join group it must be the last row of the group; otherwise an error will occur.*

Table 4.8 Options for PDF_add_table_cell( )

key	explanation
rowheight[1]	*(Float or percentage) Height of the row specified in the* `row` *parameter, specified in user coordinates or as a percentage of the table's first fitbox (see PDF_fit_table( )). If percentages are used this option is required, and all row heights must be specified as percentages.*
rowscale-group[1]	*(String) Name of a row group to which the row will be added. All rows in a group will be scaled uniformly if one of the rows in the group must be enlarged to completely hold long text. If a cell spans multiple rows the affected rows form a scale group automatically.*
rowjoin-group[1]	*(String) Name of a row group to which the row will be added. All rows in the group will be kept together in a table instance. The rows in a group must be numbered consecutively. If a cell spans multiple rows the affected rows do not automatically form a join group.*
rowspan	*(Integer) Number of rows spanned by the cell. Default: 1*
textflow	*(Textflow handle) The Textflow associated with the handle will be placed in the inner cell box. A Textflow handle can only be used once in a table, and must not be used outside that table. Default: none*

1. The last specification of this option is dominant; earlier specifications for the same row or column will be ignored.

C++ Java **String fit_table(int table, double llx, double lly, double urx, double ury, String optlist)**

Perl PHP **string PDF_fit_table(resource p, int table, float llx, float lly, float urx, float ury, string optlist)**

C **const char *PDF_fit_table(PDF *p,**
int table, double llx, double lly, double urx, double ury, const char *optlist)

Fully or partially place a table on the page.

table A valid table handle retrieved with a call to *PDF_add_table_cell( )*.

llx, lly, urx, ury Coordinates of the lower left and upper right corners of the target rectangle for the table instance (the fitbox) in user coordinates. The corners can also be specified in reverse order.

optlist An option list specifying filling details according to Table 4.9. The following options can be used:

- General options: *blind, errorpolicy, horshrinklimit, rewind, vertshrinklimit*
- Table contents: *fill, header, footer, stroke*
- Visualization aids for development or debugging: *showborder, showcells, showgrid*

Returns A string which specifies the reason for returning from the function:

- *_stop:* all rows in the table have been processed.
- *_boxfull:* there are still rows to be placed, but not enough space is available in the table's fitbox; another call to *PDF_fit_table( )* is required for processing the remaining rows.
- *_error:* an error occurred; call *PDF_get_errmsg( )* to obtain details about the problem.
- Any other string: the string supplied to the *return* option in a call to *PDF_add_table_cell( )*.

Details Place the table on the page. The table cells must have been filled with prior calls to *PDF_add_table_cell( )*. If the full table doesn't fit on the page, a table instance is placed; one or more instances can be placed with subsequent calls to this function depending on the return value. The contents of a table cell will be placed in the following order:

- Shading: the areas specified with the *fill* option will be filled in the following order: *table, colother, colodd, coleven, col#, collast, rowother, rowodd, roweven, row#, rowlast, header, footer.*
- Matchbox shading: single cell areas which are defined by a *matchbox* definition.
- Contents: the specified cell contents will be placed in the following order: image, imported PDF page, Textflow, textline.
- Matchbox ruling: single cell areas which are defined by a *matchbox* definition.
- Ruling: the lines specified with the *stroke* option will be stroked with *linecap=projecting* and *linejoin=miter* in the following order: *other, horother, hor#, horlast, vertother, vert#, vertlast, frame*. Cells which span multiple rows or columns will not be disturbed by strokes. Similarly, lines will not be stroked around cells with a *matchbox* which specifies border decoration (unless the matchbox uses the inner cell box). The table border lines *verto, horo, vertN, and horN* will be suppressed if *frame* is specified.
- Named matchboxes: these can be filled with other elements like annotations, form fields, images etc. outside of the table functions.

Scope *page, pattern, template, glyph*

Table 4.9 Options for PDF_fit_table( )

key	explanation
blind	*(Boolean) If* `true`*, all calculations will be performed, but no output will be created. Default:* `false`
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
fill	*(List of option lists) This option can be used to fill rows or columns with color (the* `matchbox` *option can be used to fill single cells with color, see Section 4.4, »Matchboxes«, page 81). The following suboptions are supported:* **area** *(Keyword) Table area(s) to be filled:* **col#** *column number # in the table* **collast** *last column* **coleven** *all columns with even numbers (according to* `col` *in PDF_add_table_cell( ))* **colodd** *all columns with odd numbers* **colother** *all other columns* **row#** *row number # in the table* **rowlast** *last body row in the table instance* **roweven** *all rows with even numbers (according to* `row` *in PDF_add_table_cell( ))* **rowodd** *all rows with odd numbers* **header** *all rows in the header group* **footer** *all rows in the footer group* **rowother** *all other body rows* **table** *complete table area (i.e. all rows in the table)* **fillcolor** *(Color; required) Color for the fill area* *Examples:* *fill all rows in the table with red:* `fill = { {area=table fillcolor={rgb 1 0 0}} }` *fill odd-numbered rows with green and even-numbered rows with red:* `fill = { {area=rowodd fillcolor={rgb 0 1 0}} {area=roweven fillcolor={rgb 1 0 0}} }`
footer	*(Integer) Number of final (footer) rows in the table definition which will be repeated at the bottom of the table instance. Default: 0 (no footer rows)*
header	*(Integer) Number of initial (header) rows in the table definition which will be repeated at the top of the table instance. Default: 0 (no header rows)*
horshrinklimit	*(Float or percentage) Lower limit for the horizontal shrinking factor which will be used when the table is shrunk to fit in the table's fitbox (if a percentage is supplied) or the absolute difference between the table width and the width of the fitbox (if a float is supplied). Default: 50%*

Table 4.9 Options for PDF_fit_table( )

key	explanation
rewind	*(Integer: -1, 0, or 1) State of the table is reset to the state before some other call to PDF_fit_table( ). Currently the following values are supported. Default: 0.* *1* Rewind to the state before the first call to PDF_fit_table( ). *0* Don't reset the table. *-1* Rewind to the state before the last call to PDF_fit_table( ).
showborder	*(Boolean) If* `true`*, the outer border of the table will be stroked using the current graphics state. Default:* `false`
showcells	*(Boolean) If* `true`*, the border of each inner cell box will be stroked using the current graphics state. Default:* `false`
showgrid	*(Boolean) If* `true`*, the vertical and horizontal boundary of all columns and rows will be stroked. Default:* `false`
stroke	*(List of option lists) This option can be used to create stroked lines at the cell borders. The following suboptions are supported:* **line** *(Keyword) Table line(s) to be stroked:* **vert#** *vertical line at the right border of column number #;* `vert0` *is the left table border* **vertfirst** *first vertical line (equivalent to* `vert0`*)* **vertlast** *last vertical line* **vertother** *all other vertical lines* **hor#** *horizontal line at the bottom of row number # in the table;* `row0` *is the top border* **horfirst** *first horizontal line in the table instance* **horother** *all other horizontal lines* **horlast** *last horizontal line in the table instance* **frame** *outer border of the table* **other** *all unspecified lines* **linewidth** *(Float) Line width, where 0 means no line. Default: 1* **strokecolor** *(Color) Line color. Default: black* **dasharray** *(List of floats) 0-8 values for defining a dash pattern (see PDF_setdashpattern( )). Default: { } (empty list, i.e. stroked line)* **dashphase** *(Float) Distance into the dash pattern at which to start the dash (see PDF_setdashpattern( )). Default: 0* *Examples:* *stroke all lines with black and linewidth 1:* `stroke = {line=other}` *stroke the outer border lines with linewidth 0.5:* `stroke = { {line=frame linewidth=0.5} }` *stroke the outer border lines with linewidth 0.5, and all other lines with linewidth 0.1:* `stroke = { {line=frame linewidth=0.5}  {line=other linewidth=0.1} }`
vertshrinklimit	*(Float or percentage) The lower limit for the vertical shrinking factor which will be used when the table is shrunk to fit the table's fitbox (if a percentage is supplied) or the absolute difference between the height of the table instance and the height of the fitbox (if a float is supplied). Default: 90%*

C++ Java ***double info_table(int table, String keyword)***

Perl PHP ***float PDF_info_table(resource p, int table, string keyword)***

C ***double PDF_info_table(PDF *p, int table, const char *keyword)***

Retrieve table information related to the most recently placed table instance.

table A valid table handle retrieved with a call to *PDF_add_table_cell( )*. The table handle must already have been used in at least one call to *PDF_fit_table( )* since the returned values are meaningful only after placing a table instance on the page.

keyword A keyword specifying the requested information according to Table 4.10.

Returns The value of some table parameter as requested by *keyword*. This function returns correct geometry information even in blind mode.

Scope any except *object*

Table 4.10 Keywords for PDF_info_table( )

keyword	explanation
firstbodyrow	*Number of the first body row in the most recently placed table instance*
height	*Height of the table instance*
horboxgap	*Difference between the width of the table instance and the width of the fitbox. If the table had to be shrunk the value will specify the deviation from the width of the fitbox (i.e. a negative value).*
horshrinking	*Horizontal shrinking factor as a percentage of the calculated table width. If the table had to be shrunk horizontally the value will specify the shrinking percentage, ortherwise it will be 100.*
lastbodyrow	*Number of the last body row in the most recently placed table instance*
returnreason	*String index of the return reason*
rowcount	*Number of rows in the most recently placed table instance (including headers and footers)*
rowsplit	*1 if the last row had to be split, 0 otherwise*
vertboxgap	*Difference between the height of the table instance and the height of the fitbox. If the table had to be shrunk the value will specify the deviation from the height of the fitbox (i.e. a negative value).*
verticalgap	*Height of the vertical gap between the most recently generated table instance and the fitbox.*
vert-shrinking	*Vertical shrinking factor as a percentage of the calculated table height. If the table had to be shrunk vertically the value will specify the shrinking percentage, ortherwise it will be 100.*
width	*Width of the table instance*
x1, y1, ... , x4, y4	*Coordinates of the corners of the table box in user coordinates, counterclockwise starting at the lower left corner*
xvertline#	*x coordinate of the vertical line with number #.* `xvertline0` *is the left table border.*
yhorline#	*x coordinate of the horizontal line with number #.* `yhorline0` *is the top table border.*

C++ Java ***void delete_table(int table, String optlist)***

Perl PHP ***PDF_delete_table(resource p, int table, string optlist)***

C ***void PDF_delete_table(PDF *p, int table, const char *optlist)***

Delete a table and all associated data structures.

table A valid table handle retrieved with a call to *PDF_add_table_cell( )*.

optlist An option list specifying cleanup options according to Table 4.11.

Details Tables which have not been deleted with this function will be deleted automatically at the end of the enclosing *document* scope.

Scope any

Table 4.11 Options for PDF_delete_table( )

key	***explanation***
keephandles	*(Boolean) If* `false`*, all handles supplied to the* `textflow`*,* `image`*, and* `pdipage` *options of PDF_add_table_cell( ) will automatically be deleted. Default:* `false`

4.4 Matchboxes

Matchboxes are not defined with a dedicated function, but with the *matchbox* option in the function call which creates the actual element:

- text lines: *PDF_fit_textline( )*, *PDF_fill_textblock( )* with the *textflow* property set to *false*
- Textflow fragments: *PDF_create_textflow( )*, *PDF_add_textflow( )*, *PDF_fill_textblock( )* with the *textflow* property set to *true*
- imported PDF pages: *PDF_fit_pdi_page( )*, *PDF_fill_pdf_block( )*
- images and templates: *PDF_fit_image( )*, *PDF_fill_image_block( )*
- table cells: *PDF_add_table_cell( )*

Matchboxes are defined with the *matchbox* option of these functions. It expects an option list which supports the suboptions listed in Table 4.12. Details of the rectangles corresponding to a matchbox can be queried with *PDF_info_matchbox( )*.

Table 4.12 Suboptions for the `matchbox` *option of various functions*

option	explanation
fillcolor	*(Color) Fill color for the rectangle. Default:* `none`
strokecolor	*(Color) Stroke color for the rectangle's border. Default: black*
borderwidth	*(Float) Line width for the rectangle's border. Default: 0*
boxheight	*(List of two keywords; only for textline and Textflow) Defines the vertical extent of the text box. Two keywords can be specified separately for the extent above and below the baseline:* `none` *(no extent),* `xheight`, `descender`, `capheight`, `ascender`, `fontsize`, `leading` *Textflow: the values corresponding to the text at the beginning of the matchbox will be used. Default:* `{capheight none}`
boxwidth	*(Float; only for Textflow) Width of the matchbox. If this option is supplied, horizontal space of the specified width will be inserted between the* `matchbox` *option and the next text fragment or the* `matchbox end` *specification. This may be useful to reserve space for inserting an image, template, or PDF page in the Textflow. Default: 0*
clipping	*(List of 4 floats or 4 percentages; only for images and imported PDF pages; will be ignored if the* `innerbox` *option has been specified) Coordinates of the lower left and upper right corner of a rectangle within the image or page specifying which part should be displayed. With images, the clipping rectangle can be specified in pixels or as a percentage of the width/height. With PDF pages, the clipping rectangle can be specified in default units or as a percentage of the width/height of the page's crop box. Default: {0% 0% 100% 100%}*
dasharray	*(Float list) List of 0-8 values for defining a dash pattern for the rectangle border in user coordinates (see PDF_setdashpattern( )). Default: { }*
dashphase	*(Float) Distance into the dash pattern at which to start the dash for the rectangle's border (see PDF_setdashpattern( )). Default: 0*
drawleft drawbottom drawright drawtop	*(Boolean) If* `true`*, the corresponding border of the rectangle will be drawn. Default:* `true`
end	*(Boolean; only for Textflow) Specifies the end of the matchbox. If* `true`*, all other* `matchbox` *options will be ignored. Matchboxes in Textflows cannot be nested. The width of a Textflow matchbox is defined by the option* `boxwidth` *(if specified) and the extent of the text enclosed in the options* `matchbox` *and* `matchbox=end`*. If the* `end` *option has not been specified, the matchbox will end after the last character in the Textflow.*

Table 4.12 Suboptions for the matchbox *option of various functions*

option	*explanation*
innerbox	*(Boolean; only for table cells, and TIFF and JPEG images) Table cells: If* true*, the box will be reduced by the margins defined for the cell; otherwise the full box will be used.* *TIFF and JPEG images: If the image contains a clipping path the bounding box of the clipping path will be used instead of the full image.* *Default:* false
linecap	*(Integer or keyword) Shape at the end of a path (see PDF_setlinecap( )); must be 0, 1, or 2, or one of the corresponding keywords* butt*,* round*, or* projecting*. Default:* butt
linejoin	*(Integer or keyword) Shape at the corners of paths (see PDF_setlinejoin( )); must be 0, 1, or 2, or one of the corresponding keywords* miter*,* round*, or* bevel*. Default:* miter
margin	*(Float or percentage) Additional margin for the matchbox rectangle, specified in user coordinates (must be greater than or equal to 0) or as a percentage of the rectangle width or height (must be less than 100%). Default: 0*
name	*(Name string) Name of the matchbox which can be used in PDF_info_matchbox( ). If the name has already been assigned to a matchbox, another rectangle for this name will be created. Matchbox names can be used until the end of the current page. Default: no name*
offsetleft ***offsetbottom*** ***offsetright*** ***offsettop***	*(Float or percentage) User-defined offset from the left/right/bottom/top edge of the calculated initial rectangle and the desired box. The values are specified in user coordinates or as a percentage of the rectangle's width (for* offsetleft/offsetright*) or height (for* offsetbottom/offsettop*). Negative values are allowed, and can be used to extend the matchbox. Default of* offsetleft/offsetbottom*:* margin*; Default of* offsetright/offsettop*:* -margin
openrect	*(Boolean; only for Textflow and table cells) Textflow: if a matchbox rectangle is split to the next line, the right border of the first rectangle and the left border of the second rectangle will not be drawn. Table cells: If a table row is split to the next table instance the bottom border of the first part and the top border of the second part will not be drawn. Default:* false

C++ Java double info_matchbox(String boxname, int num, String keyword)

Perl PHP float PDF_info_matchbox(resource p, string boxname, int num, string keyword)

C double PDF_info_matchbox(PDF *p,const char *boxname, int len, int num, const char *keyword)

Query information about a matchbox on the current page.

boxname (Name string) Name of the matchbox. The name must have been defined with the *name* suboption of the *matchbox* option when the matchbox was defined.

len (C language binding only) Length of *name* in bytes for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

num Number of the requested matchbox rectangle (the first has number 1).

keyword A keyword specifying the requested information according to Table 4.13.

Returns The value of some matchbox parameter as requested by *keyword*. If a matchbox with the specified name or number does not exist on the current page all keywords will return the value 0.

Scope *page, pattern, template, glpyh, path, font*

Table 4.13 Keywords for PDF_info_matchbox( )

keyword	***explanation***
count	*(The* num *parameter will be ignored) Number of rectangles*
exists	*1 if the rectangle exists, 0 otherwise*
height	*Height of the rectangle in user coordinates*
width	*Width of the rectangle in user coordinates*
x1, y1, ... , x4, y4	*Position of the i-th rectangle corner (i=1, 2, 3, 4) in user coordinates. In the coordinate system of the respective fit element (image, text, etc.),* x1, y1 *correspond to the upper left,* x2, y2 *to the lower left,* x3, y3 *to the lower right and* x4, y4 *to the upper right corner*

5 Graphics Functions

5.1 Graphics State

All graphics state parameters are restored to their default values at the beginning of a page. The default values are documented in the respective function descriptions. Functions related to the text state are listed in Chapter 3, »Text Functions«, page 35.

Note *None of the graphics state functions must be used in path scope.*

C++ Java ***void setdash(double b, double w)***

Perl PHP ***PDF_setdash(resource p, float b, float w)***

C ***void PDF_setdash(PDF *p, double b, double w)***

Set the current dash pattern.

b, w The number of alternating black and white units. *b* and *w* must be non-negative numbers.

Details In order to produce a solid line, set *b=w=0*. The *dash* parameter is set to solid at the beginning of each page.

Scope *page, pattern, template, glyph*

C++ Java ***void setdashpattern(String optlist)***

Perl PHP ***PDF_setdashpattern(resource p, string optlist)***

C ***void PDF_setdashpattern(PDF *p, const char *optlist)***

Set a dash pattern defined by an option list.

optlist An option list according to Table 5.1. An empty list will generate a solid line. The following options can be used:
dasharray, dashphase

Table 5.1 Options for PDF_setdashpattern( )

option	description
dasharray	*(List of floats) List of 2-8 alternating values for the lengths of dashes and gaps for stroked paths (measured in the user coordinate system). The array values must be greater than zero. They will be cyclically reused until the complete path is stroked.*
dashphase	*(Float) Distance into the dash pattern at which to start the dash. Default: 0*

Details The *dash* parameter is set to a solid line at the beginning of each page.

Scope *page, pattern, template, glyph*

C++ Java ***void setflat(double flatness)***
Perl PHP ***PDF_setflat(resource p, float flatness)***
C ***void PDF_setflat(PDF *p, double flatness)***

Set the *flatness* tolerance.

flatness A positive number which describes the maximum distance (in device pixels) between the path and an approximation constructed from straight line segments.

Details The *flatness* tolerance is set to the default value of 1 at the beginning of each page.

Scope *page, pattern, template, glyph*

C++ Java ***void setlinejoin(int linejoin)***
Perl PHP ***PDF_setlinejoin(resource p, int linejoin)***
C ***void PDF_setlinejoin(PDF *p, int linejoin)***

Set the *linejoin* style.

linejoin Specifies the shape at the corners of paths that are stroked, see Table 5.2.

Details The *linejoin* style is set to the default value of 0 at the beginning of each page.

Scope *page, pattern, template, glyph*

Table 5.2 Values of the `linejoin` *style*

value	*description (from the PDF reference)*	*examples*
0	*Miter joins: the outer edges of the strokes for the two segments are continued until they meet. If the extension projects too far, as determined by the miter limit, a bevel join is used instead.*	
1	*Round joins: a circular arc with a diameter equal to the line width is drawn around the point where the segments meet and filled in, producing a rounded corner.*	
2	*Bevel joins: the two path segments are drawn with butt end caps (see the discussion of the* `linecap` *parameter), and the resulting notch beyond the ends of the segments is filled in with a triangle.*	

C++ Java ***void setlinecap(int linecap)***
Perl PHP ***PDF_setlinecap(resource p, int linecap)***
C ***void PDF_setlinecap(PDF *p, int linecap)***

Set the *linecap* parameter.

linecap Controls the shape at the end of a path with respect to stroking, see Table 5.3.

Details The *linecap* parameter is set to the default value of 0 at the beginning of each page.

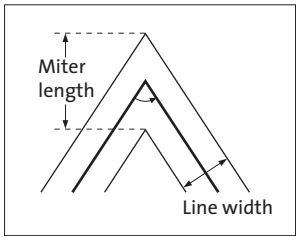
Scope *page, pattern, template, glyph*

Table 5.3 Values of the `linecap` *parameter*

value	*description (from the PDF reference)*	*examples*
0	*Butt end caps: the stroke is squared off at the endpoint of the path.*	
1	*Round end caps: a semicircular arc with a diameter equal to the line width is drawn around the endpoint and filled in.*	
2	*Projecting square end caps: the stroke extends beyond the end of the line by a distance which is half the line width and is squared off.*	

C++ Java ***void setmiterlimit(double miter)***
Perl PHP ***PDF_setmiterlimit(resource p, float miter)***
C ***void PDF_setmiterlimit(PDF *p, double miter)***

Set the miter limit.



miter A value greater than or equal to 1 which controls the spike produced by miter joins.

Details If the *linejoin* style is set to 0 (miter join), two line segments joining at a small angle will result in a sharp spike. This spike will be replaced by a straight end (i.e. the miter join will be changed to a bevel join) when the ratio of the miter length and the line width exceeds the miter limit. The miter limit is set to the default value of 10 at the beginning of each page. This corresponds to an angle of roughly 11.5 degrees.

Scope *page, pattern, template, glyph*

C++ Java ***void setlinewidth(double width)***
Perl PHP ***PDF_setlinewidth(resource p, float width)***
C ***void PDF_setlinewidth(PDF *p, double width)***

Set the current line width.

width The line width in units of the current user coordinate system.

Details The *width* parameter is set to the default value of 1 at the beginning of each page.

Scope *page, pattern, template, glyph*

C++ Java ***void initgraphics( )***
Perl PHP ***PDF_initgraphics(resource p)***
C ***void PDF_initgraphics(PDF *p)***

Reset all color and graphics state parameters to their defaults.

Details The fill and stroke colors, line width, line cap style, line join style, miter limit, dash pattern, and flatness tolerance settings, and the coordinate system (but not the text state

parameters) are reset to their respective defaults. The current clipping path is not affected.

This function may be useful in situations where the program flow doesn't allow for easy use of *PDF_save( )/PDF_restore( )*.

Scope *page, pattern, template, glyph*

5.2 Saving and Restoring Graphics States

C++ Java ***void save( )***
Perl PHP ***PDF_save(resource p)***
C ***void PDF_save(PDF *p)***

Save the current graphics state to a stack.

Details The graphics state contains parameters that control all types of graphics objects. Saving the graphics state is not required by PDF; it is only necessary if the application wishes to return to some specific graphics state later (e.g. a custom coordinate system) without setting all relevant parameters explicitly again. The following items are subject to save/restore:

- graphics parameters which have been set with the corresponding functions: clipping path, coordinate system, current point, flatness tolerance, line cap style, dash pattern, line join style, line width, miter limit;
- color parameters: fill and stroke colors;
- graphics parameters which have been set with explicit graphics states in *PDF_set_gstate( )* (see Section 5.4, »Explicit Graphics States«, page 90);
- text position and the following text-related parameters: *charspacing*, *wordspacing*, *horizscaling*, *italicangle*, *leading*, *font*, *fontsize*, *textrendering*, *textrise*.

Pairs of *PDF_save( )* and *PDF_restore( )* may be nested. Although the PDF specification doesn't limit the nesting level of save/restore pairs, applications must keep the nesting level below 26 in order to avoid printing problems caused by restrictions in the PostScript output produced by PDF viewers, and to allow for additional save levels required by PDFlib internally.

Scope *page, pattern, template, glyph;* must always be paired with a matching *PDF_restore( )* call. *PDF_save( )* and *PDF_restore( )* calls must be balanced on each page, pattern, template, and glyph description.

Params Most text-related parameters are affected by save/restore; see list above. The following parameters are not subject to save/restore: *fillrule*, *kerning*, *underline*, *overline*, *strikeout*.

C++ Java ***void restore( )***
Perl PHP ***PDF_restore(resource p)***
C ***void PDF_restore(PDF *p)***

Restore the most recently saved graphics state from the stack.

Details The corresponding graphics state must have been saved on the same page, pattern, or template.

Scope *page, pattern, template, glyph;* must always be paired with a matching *PDF_save( )* call. *PDF_save( )* and *PDF_restore( )* calls must be balanced on each page, pattern, template, and glyph description.

5.3 Coordinate System Transformations

All transformation functions *(PDF_translate( ), PDF_scale( ), PDF_rotate( ), PDF_skew( ), PDF_concat( ), PDF_setmatrix( )*, and *PDF_initgraphics( ))* change the coordinate system used for drawing subsequent objects. They do not affect existing objects on the page at all.

C++ Java ***void translate(double tx, double ty)***

Perl PHP ***PDF_translate(resource p, float tx, float ty)***

C ***void PDF_translate(PDF *p, double tx, double ty)***

Translate the origin of the coordinate system.

tx, ty The new origin of the coordinate system is the point *(tx, ty)*, measured in the old coordinate system.

Scope *page, pattern, template, glyph*

C++ Java ***void scale(double sx, double sy)***

Perl PHP ***PDF_scale(resource p, float sx, float sy)***

C ***void PDF_scale(PDF *p, double sx, double sy)***

Scale the coordinate system.

sx, sy Scaling factors in *x* and *y* direction.

Details This function scales the coordinate system by *sx* and *sy*. It may also be used for achieving a reflection (mirroring) by using a negative scaling factor. One unit in the *x* direction in the new coordinate system equals *sx* units in the *x* direction in the old coordinate system; analogous for *y* coordinates.

Scope *page, pattern, template, glyph*

C++ Java ***void rotate(double phi)***

Perl PHP ***PDF_rotate(resource p, float phi)***

C ***void PDF_rotate(PDF *p, double phi)***

Rotate the coordinate system.

phi The rotation angle in degrees.

Details Angles are measured counterclockwise from the positive x axis of the current coordinate system. The new coordinate axes result from rotating the old coordinate axes by *phi* degrees.

Scope *page, pattern, template, glyph*

C++ Java ***void skew(double alpha, double beta)***

Perl PHP ***PDF_skew(resource p, float alpha, float beta)***

C ***void PDF_skew(PDF *p, double alpha, double beta)***

Skew the coordinate system.

alpha, beta Skewing angles in *x* and *y* direction in degrees.

Details Skewing (or shearing) distorts the coordinate system by the given angles in *x* and *y* direction. *alpha* is measured counterclockwise from the positive *x* axis of the current coordinate system, *beta* is measured clockwise from the positive *y* axis. Both angles must be in the range -360° < *alpha, beta* < 360°, and must be different from -270°, -90°, 90°, and 270°.

Scope *page, pattern, template, glyph*

C++ Java ***void concat(double a, double b, double c, double d, double e, double f)***

Perl PHP ***PDF_concat(resource p, float a, float b, float c, float d, float e, float f)***

C ***void PDF_concat(PDF *p, double a, double b, double c, double d, double e, double f)***

Apply a transformation matrix to the current coordinate system.

a, b, c, d, e, f Elements of a transformation matrix. The six values make up a matrix in the same way as in PostScript and PDF (see references). In order to avoid degenerate transformations, *a*d* must not be equal to *b*c*.

Details This function applies a matrix to the current coordinate system. It allows for the most general form of transformations. Unless you are familiar with the use of transformation matrices, the use of *PDF_translate( ), PDF_scale( ), PDF_rotate( ),* and *PDF_skew( )* is suggested instead of this function. The coordinate system is reset to the default coordinate system (i.e. the current transformation matrix is the identity matrix *[1, 0, 0, 1, 0, 0])* at the beginning of each page.

Scope *page, pattern, template, glyph*

C++ Java ***void setmatrix(double a, double b, double c, double d, double e, double f)***

Perl PHP ***PDF_setmatrix(resource p, float a, float b, float c, float d, float e, float f)***

C ***void PDF_setmatrix(PDF *p, double a, double b, double c, double d, double e, double f)***

Explicitly set the current transformation matrix.

a, b, c, d, e, f See *PDF_concat( )*.

Details This function is similar to *PDF_concat( )*. However, it disposes of the current transformation matrix, and completely replaces it with the new matrix.

Scope *page, pattern, template, glyph*

5.4 Explicit Graphics States

C++ Java ***int create_gstate(String optlist)***
Perl PHP ***int PDF_create_gstate(resource p, string optlist)***
C ***int PDF_create_gstate(PDF *p, const char *optlist)***

Create a graphics state object subject to various options.

optlist An option list containing options for graphics states according to Table 5.4. The following options can be used:
alphaisshape, blendmode, flatness, linecap, linejoin, linewidth, miterlimit, opacityfill, opacitystroke, overprintfill, overprintmode, overprintstroke, renderingintent, smoothness, strokeadjust, textknockout

Returns A graphics state handle that can be used in subsequent calls to *PDF_set_gstate( )* during the enclosing *document* scope.

Details The option list may contain any number of graphics state parameters. Not all parameters are allowed for all PDF versions. The table lists the minimum required PDF version.

Scope *document, page, pattern, template, glyph*

Table 5.4 Options for PDF_create_gstate( )

key	explanation and possible values
alphaisshape	*(Boolean; PDF 1.4) Sources of alpha are treated as shape (true) or opacity (false). Default:* `false`
blendmode	*(Keyword list; PDF 1.4; if used in PDF/A mode it must have the value* `Normal`*) Name of the blend mode. Multiple blend modes can be specified. Possible values:* `Color`, `ColorDodge`, `ColorBurn`, `Darken`, `Difference`, `Exclusion`, `HardLight`, `Hue`, `Lighten`, `Luminosity`, `Multiply`, `None`, `Normal`, `Overlay`, `Saturation`, `Screen`, `SoftLight`. *Default:* `None`
flatness	*(Float) Maximum distance between a path and its approximation (see PDF_setflat( )), must be > 0*
linecap	*(Integer or keyword) Shape at the end of a path (see PDF_setlinecap( )); must be 0, 1, or 2, or one of the corresponding keywords* `butt`, `round`, `projecting`
linejoin	*(Integer or keyword) Shape at the corners of paths (see PDF_setlinejoin( )); must be 0, 1, or 2, or one of the corresponding keywords* `miter`, `round`, `bevel`
linewidth	*(Float) Line width (see PDF_setlinewidth( )); must be > 0*
miterlimit	*(Float) Controls the spike produced by miter joins, which must be >= 1 (see PDF_setmiterlimit( ))*
opacityfill	*(Float; PDF 1.4; if used in PDF/A mode it must have the value 1) Constant alpha for fill operations; must be >= 0 and <= 1.*
opacitystroke	*(Float; PDF 1.4; if used in PDF/A mode it must have the value 1) Constant alpha for stroke operations; must be >=0 and <=1*
overprintfill	*(Boolean) Overprint for operations other than stroke. Default:* `false`
overprintmode	*(Integer) Overprint mode. 0 means that each color component replaces previously placed marks; mode 1 (called »overprinting default is nonzero overprinting« in Acrobat) means that a color component of 0 leaves the corresponding component unchanged. Default:* `0`
overprintstroke	*(Boolean) Overprint for stroke operations. Default:* `false`

Table 5.4 Options for PDF_create_gstate( )

key	explanation and possible values
renderingintent	*(Keyword) Color rendering intent used for gamut compression; possible keywords:* `Auto`, `AbsoluteColorimetric`, `RelativeColorimetric`, `Saturation`, `Perceptual`
smoothness	*(Float) Maximum error of a linear interpolation for a shading; must be >= 0 and <= 1*
strokeadjust	*(Boolean) Whether or not to apply automatic stroke adjustment. Default:* `false`
textknockout	*(Boolean; PDF 1.4) With respect to compositing, glyphs in a text object will be treated as separate objects* (`false`) *or as a single object* (`true`). *Default:* `true`

C++ Java ***void set_gstate(int gstate)***

Perl PHP ***PDF_set_gstate(resource p, int gstate)***

C ***void PDF_set_gstate(PDF *p, int gstate)***

Activate a graphics state object.

gstate A handle for a graphics state object retrieved with *PDF_create_gstate( )*.

Details All options contained in the graphics state object will be set. Graphics state options accumulate when this function is called multiply. Options which are not explicitly set in the graphics state object will keep their current values. All graphics state options will be reset to their default values at the beginning of a page.

Scope *page, pattern, template, glyph*

5.5 Path Construction

Table 5.5 lists relevant value key names for this section.

Table 5.5 Keys for PDF_get_value( ) (see Section 2.1, »Parameter Handling«, page 11)

key	explanation
currentx[1] ***currenty***[1]	*The x or y coordinate (in units of the current coordinate system), respectively, of the current point. Scope: page, pattern, template, path*
ctm_a[1] ***ctm_b***[1] ***ctm_c***[1] ***ctm_d***[1] ***ctm_e***[1] ***ctm_f***[1]	*The components of the current transformation matrix (CTM) for vector graphics. Scope: page, pattern, template, path*

1. Cannot be used with PDF_set_value( ) since there are corresponding API functions available for setting these values

Note *Make sure to call one of the functions in Section 5.6, »Path Painting and Clipping«, page 95, after using the functions in this section, or the constructed path will have no effect, and subsequent operations may raise an exception.*

C++ Java ***void moveto(double x, double y)***

Perl PHP ***PDF_moveto(resource p, float x, float y)***

C ***void PDF_moveto(PDF *p, double x, double y)***

Set the current point for graphics output.

x, y The coordinates of the new current point.

Details The current point is set to the default value of *undefined* at the beginning of each page. The current points for graphics and the current text position are maintained separately.

Scope *page, pattern, template, path, glyph;* this function starts *path* scope.

Params *currentx, currenty*

C++ Java ***void lineto(double x, double y)***

Perl PHP ***PDF_lineto(resource p, float x, float y)***

C ***void PDF_lineto(PDF *p, double x, double y)***

Draw a line from the current point to another point.

x, y The coordinates of the second endpoint of the line.

Details This function adds a straight line from the current point to *(x, y)* to the current path. The current point must be set before using this function. The point *(x, y)* becomes the new current point.

The line will be centered around the »ideal« line, i.e. half of the linewidth (as determined by the value of the *linewidth* parameter) will be painted on each side of the line connecting both endpoints. The behavior at the endpoints is determined by the value of the *linecap* parameter.

Scope *path*

Params *currentx, currenty*

C++ Java ***void curveto(double x1, double y1, double x2, double y2, double x3, double y3)***

Perl PHP ***PDF_curveto(resource p, float x1, float y1, float x2, float y2, float x3, float y3)***

C ***void PDF_curveto(PDF *p, double x1, double y1, double x2, double y2, double x3, double y3)***

Draw a Bézier curve from the current point, using three more control points.

x1, y1, x2, y2, x3, y3 The coordinates of three control points.

Details A Bézier curve is added to the current path from the current point to *(x3, y3)*, using *(x1, y1)* and *(x2, y2)* as control points. The current point must be set before using this function. The endpoint of the curve becomes the new current point.

Scope *path*

Params *currentx, currenty*

C++ Java ***void circle(double x, double y, double r)***

Perl PHP ***PDF_circle(resource p, float x, float y, float r)***

C ***void PDF_circle(PDF *p, double x, double y, double r)***

Draw a circle.

x, y The coordinates of the center of the circle.

r The radius of the circle.

Details This function adds a circle to the current path as a complete subpath. The point *(x + r, y)* becomes the new current point. The resulting shape will be circular in user coordinates. If the coordinate system has been scaled differently in *x* and *y* directions, the resulting curve will be elliptical.

Scope *page, pattern, template, path, glyph;* this function starts *path* scope.

Params *currentx, currenty*

C++ Java ***void arc(double x, double y, double r, double alpha, double beta)***

Perl PHP ***PDF_arc(resource p, float x, float y, float r, float alpha, float beta)***

C ***void PDF_arc(PDF *p, double x, double y, double r, double alpha, double beta)***

Draw a counterclockwise circular arc segment.

x, y The coordinates of the center of the circular arc segment.

r The radius of the circular arc segment. *r* must be nonnegative.

alpha, beta The start and end angles of the circular arc segment in degrees.

Details This function adds a counterclockwise circular arc segment to the current path, extending from *alpha* to *beta* degrees. For both *PDF_arc( )* and *PDF_arcn( )*, angles are measured counterclockwise from the positive x axis of the current coordinate system. If there is a current point an additional straight line is drawn from the current point to the starting point of the arc. The endpoint of the arc becomes the new current point.

The arc segment will be circular in user coordinates. If the coordinate system has been scaled differently in *x* and *y* directions the resulting curve will be elliptical.

Scope *page, pattern, template, path, glyph;* this function starts *path* scope.

Params *currentx, currenty*

C++ Java ***void arcn(double x, double y, double r, double alpha, double beta)***

Perl PHP ***PDF_arcn(resource p, float x, float y, float r, float alpha, float beta)***

C ***void PDF_arcn(PDF *p, double x, double y, double r, double alpha, double beta)***

Draw a clockwise circular arc segment.

Details Except for the drawing direction, this function behave exactly like *PDF_arc( )*. In particular, the angles are still measured *counterclockwise* from the positive *x* axis.

C++ Java ***void rect(double x, double y, double width, double height)***

Perl PHP ***PDF_rect(resource p, float x, float y, float width, float height)***

C ***void PDF_rect(PDF *p, double x, double y, double width, double height)***

Draw a rectangle.

x, y The coordinates of the lower left corner of the rectangle.

width, height The size of the rectangle.

Details This function adds a rectangle to the current path as a complete subpath. Setting the current point is not required before using this function. The point *(x, y)* becomes the new current point. The lines will be centered around the »ideal« line, i.e. half of the linewidth (as determined by the value of the *linewidth* parameter) will be painted on each side of the line connecting the respective endpoints.

Scope *page, pattern, template, path, glyph;* this function starts *path* scope.

Params *currentx, currenty*

C++ Java ***void closepath( )***

Perl PHP ***PDF_closepath(resource p)***

C ***void PDF_closepath(PDF *p)***

Close the current path.

Details This function closes the current subpath, i.e. adds a line from the current point to the starting point of the subpath.

Scope *path*

Params *currentx, currenty*

5.6 Path Painting and Clipping

Table 5.6 lists relevant parameter key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 5.6 Path-related keys for PDF_get/set_parameter( )

key	explanation
fillrule	*Set the current fill rule to* `winding` *or* `evenodd`*. The fill rule is used by PDF viewers to determine the interior of shapes for the purpose of filling or clipping. Since both algorithms yield the same result for simple shapes, most applications won't have to change the fill rule. The fill rule is reset to the default of winding at the beginning of each page. Scope: page, pattern, template, glyph.*

Note *Most functions in this section clear the path, and leave the current point undefined. Subsequent drawing operations must therefore explicitly set the current point (e.g. using PDF_moveto( )) after one of these functions has been called.*

C++ Java ***void stroke( )***

Perl PHP ***PDF_stroke(resource p)***

C ***void PDF_stroke(PDF *p)***

Stroke the path with the current line width and current stroke color, and clear it.

Scope *path;* this function terminates *path* scope.

C++ Java ***void closepath_stroke( )***

Perl PHP ***PDF_closepath_stroke(resource p)***

C ***void PDF_closepath_stroke(PDF *p)***

Close the path, and stroke it.

Details This function closes the current subpath (adds a straight line segment from the current point to the starting point of the path), and strokes the complete current path with the current line width and the current stroke color.

Scope *path;* this function terminates *path* scope.

C++ Java ***void fill( )***

Perl PHP ***PDF_fill(resource p)***

C ***void PDF_fill(PDF *p)***

Fill the interior of the path with the current fill color.

Details This function fills the interior of the current path with the current fill color. The interior of the path is determined by one of two algorithms (see the *fillrule* parameter). Open paths are implicitly closed before being filled.

Scope *path;* this function terminates *path* scope.

Params *fillrule*

C++ Java ***void fill_stroke( )***
Perl PHP ***PDF_fill_stroke(resource p)***
C ***void PDF_fill_stroke(PDF *p)***

Fill and stroke the path with the current fill and stroke color.

Scope *path;* this function terminates *path* scope.

Params *fillrule*

C++ Java ***void closepath_fill_stroke( )***
Perl PHP ***PDF_closepath_fill_stroke(resource p)***
C ***void PDF_closepath_fill_stroke(PDF *p)***

Close the path, fill, and stroke it.

Details This function closes the current subpath (adds a straight line segment from the current point to the starting point of the path), and fills and strokes the complete current path.

Scope *path;* this function terminates *path* scope.

Params *fillrule*

C++ Java ***void clip( )***
Perl PHP ***PDF_clip(resource p)***
C ***void PDF_clip(PDF *p)***

Use the current path as clipping path, and terminate the path.

Details This function uses the intersection of the current path and the current clipping path as the clipping path for subsequent operations. The clipping path is set to the default value of the page size at the beginning of each page. The clipping path is subject to *PDF_save( )/PDF_restore( )*. It can only be enlarged by means of *PDF_save( )*/*PDF_restore( )*.

Scope *path;* this function terminates *path* scope.

C++ Java ***void endpath( )***
Perl PHP ***PDF_endpath(resource p)***
C ***void PDF_endpath(PDF *p)***

End the current path without filling or stroking it.

Details This function doesn't have any visible effect on the page. It generates an invisible path on the page.

Scope *path;* this function terminates *path* scope.

5.7 Layers

C++ Java ***int define_layer(String name, String optlist)***

Perl PHP ***int PDF_define_layer(resource p, string name, string optlist)***

C ***int PDF_define_layer(PDF *p, const char *name, int len, const char *optlist)***

Create a new layer definition (requires PDF 1.5).

name (Hypertext string) The name of the layer.

len (C language binding only) Length of *name* (in bytes) for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

optlist An option list specifying layer settings according to Table 5.7. The following options can be used:
creatorinfo, defaultstate, hypertextencoding, hypertextformat, initialexportstate, initialprintstate, initialviewstate, intent, language, onpanel, pageelement, printsubtype, zoom

Returns A layer handle which can be used in calls to *PDF_begin_layer( )* and *PDF_set_layer_dependency( )* until the end of the enclosing *document* scope.

Details PDFlib will issue a warning if a layer was defined but hasn't been used in the document.

Scope *document, page*

Table 5.7 Options for PDF_define_layer( )

option	*explanation*
creatorinfo	*(Option list) An option list describing the content and the creating application. Both of the following entries are required if this option is used:* ***creator*** *(Hypertext string) The name of the application which created the layer* ***subtype*** *(String) The type of content. Suggested values are* `Artwork` *and* `Technical`.
defaultstate	*(Boolean) Default:* `true`
hypertextencoding	*(Keyword) Specifies the encoding for the* `name` *parameter and the* `creator` *option. An empty string is equivalent to* `unicode`. *Default: the global* `hypertextencoding` *parameter*
hypertextformat	*(Keyword) Sets the format for the* `name` *parameter. Possible values are* `bytes`, `utf8`, `utf16`, `utf16le`, `utf16be`, *and* `auto`. *Default: the value of the* `hypertextformat` *parameter*
initialexportstate	*(Boolean) Specifies the layer's recommended export state. If* `true`, *Acrobat will include the layer when converting/exporting to older PDF versions or other document formats. Default:* `true`
initialprintstate	*(Boolean) The layer's recommended printing state. If* `true`, *Acrobat will include the layer when printing the document. Default:* `true`
initialviewstate	*(Boolean) The layer's recommended viewing state. If* `true`, *Acrobat will display the layer when opening the document. Default:* `true`
intent	*(Keyword) Intended use of the graphics:* `View` *or* `Design`. *Default:* `View`
language	*(Option list) Specifies the language of the layer:* ***lang*** *(String; required) The language and possibly locale in the format described in Table 2.3 for the* `lang` *option* ***preferred*** *(Boolean) If true this layer is used if there is only a partial match between the layer and the system language. Default:* `false`

Table 5.7 Options for PDF_define_layer( )

option	explanation
onpanel	*(Boolean) If* `false`*, the layer name will not be visible in Acrobat's layer panel, and therefore cannot be manipulated by the user. Default:* `true`
pageelement	*(Keyword) Specifies that the layer contains a pagination artifact: one of* `HF` *(header/footer),* `FG` *(foreground image or graphic),* `BG` *(background image or graphic), or* `L` *(logo).*
printsubtype	*(Option list) Specifies whether the layer is intended for printing:* **subtype** *(Keyword) One of* `Trapping`, `PrintersMarks`, *or* `Watermark` *specifying the kind of content in the layer.* **printstate** *(Boolean) If* `true`*, Acrobat will activate the layer contents upon printing.*
zoom	*(List of floats or percentages) One or two values specifying the layer's visibility depending on the zoom factor (1.0 means a zoom factor of 100 percent). If one value is provided, it will be used as the maximum zoom factor at which the layer should be visible; if two values are provided they specify the minimum and maximum zoom factor. The keyword* `maxzoom` *can be used to specify the largest possible zoom factor.*

C++ Java ***void set_layer_dependency(String type, String optlist)***

Perl PHP ***PDF_set_layer_dependency(resource p, string type, string optlist)***

C ***void PDF_set_layer_dependency(PDF *p, const char *type, const char *optlist)***

Define hierarchical, group, and lock conditions among layers (requires PDF 1.5).

type The type of dependency, which must be one of the following:

- *GroupAllOn:* The layer specified in the *depend* option will be visible if all layers specified in the *group* option are visible. Options specific for this type: *depend*, *group*
- *GroupAnyOn:* The layers specified in the *depend* option will be visible if any layer specified in the *group* option is visible. Options specific for this type: *depend*, *group*
- *GroupAllOff:* The layer specified in the *depend* option will be visible if all layers specified in the *group* option are invisible. Options specific for this type: *depend*, *group*
- *GroupAnyOff:* The layer specified in the *depend* option will be visible if any layer specified in the *group* option is invisible. Options specific for this type: *depend*, *group*
- *Lock:* (PDF 1.6) The layers specified in the *group* option will be locked, i.e. their state cannot be changed interactively in Acrobat. Options specific for this type: *group*
- *Parent:* Specify a hierarchical relationship between the layer specified in the *parent* option and the layers specified in the *children* option. A layer can not belong to more than one parent layer. Options specific for this type: *children, parent*
- *Radiobtn:* Specify a radiobutton relationship between the layers specified in the *group* option. This means that at most one layer in the group will be visible at a time, which is particularly useful for multiple language layers.
  Options specific for this type: *group*
- *Title:* The layer handle specified in the *parent* option does not control any page contents directly, but serves as the parent layer node for the layers specified in the *children* option. Options specific for this type: *children, parent*

optlist An option list specifying layer dependencies according to Table 5.8.

Scope *document, page*

Table 5.8 Options for PDF_set_layer_dependency( )

option	explanation
children	*(List of layer handles; only for* `type=Parent` *and* `Title`*)* *One or more layer handles specifying the layers subordinate to the provided parent layer.*
depend	*(Layer handle; only for* `type=GroupAllOn`, `GroupAnyOn`, `GroupAllOff`, *and* `GroupAnyOff`*)* *The layer which is controlled by the layers specified in the* `group` *option.*
group	*(List of layer handles; only for* `type=GroupAllOn`, `GroupAnyOn`, `GroupAllOff`, `GroupAnyOff`, `Lock`, *and* `Radiobtn`*)* *One or more layer handles comprising the group. For* `type=Lock` *all layers in the group will be locked.*
parent	*(Layer handle; only for* `type=Parent` *and* `Title`*)* *The layer which is the parent of the layers specified in the* `children` *option.*

C++ Java ***void begin_layer(int layer)***
Perl PHP ***PDF_begin_layer(resource p, int layer)***
C ***void PDF_begin_layer(PDF *p, int layer)***

Start a layer for subsequent output on the page (requires PDF 1.5).

layer The layer's handle, which must have been retrieved with *PDF_define_layer( )*.

Details All content placed on the page after this call, but before any subsequent call to *PDF_begin_layer( )* or *PDF_end_layer( )* will be part of the specified layer. The content's visibility depends on the layer's settings.

This function activates the specified layer, and deactivates any layer which may be currently active.

Layers for annotations, images, templates, and form fields can also be controlled with the *layer* option of the respective functions. Calling *PDF_define_layer( )* is not required in these cases.

Scope *page*

C++ Java ***void end_layer( )***
Perl PHP ***PDF_end_layer(resource p)***
C ***void PDF_end_layer(PDF *p)***

Deactivate all active layers (requires PDF 1.5).

Details Content placed on the page after this call will not belong to any layer. All layers must be closed at the end of a page.

In order to switch from layer A to layer B a single call to *PDF_begin_layer( )* is sufficient; it is not required to explicitly call *PDF_end_layer( )* to close layer A. *PDF_end_layer( )* is only required to create unconditional content (which is always visible), and to close all layers at the end of a page.

Scope *page*

6 Color Functions

6.1 Setting Color and Color Space

Table 6.1 lists relevant parameter key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 6.1 Color-related keys for PDF_get/set_parameter( )

key	explanation
preserveold-pantonenames	*If* `false`*, old-style Pantone spot color names will be converted to the corresponding new color names, otherwise they will be preserved. Default:* `false`*. Scope: any*
spotcolorlookup	*If* `false`*, PDFlib will not use its internal database of spot color names. This can be used to provide custom definitions of known spot colors, which may be required as a workaround to match the definitions used by other applications. This feature should be used with care, and is not recommended. Default:* `true`*. Scope: any*

Color spaces. PDFlib clients may specify the colors used for filling and stroking the interior of paths and text characters. Colors may be specified in several color spaces (each list item starts with the corresponding color space keyword for *PDF_setcolor( )* and color options):

- *gray:* Gray values between 0=black and 1=white;
- *rgb:* RGB triples, i.e. three values between 0 and 1 specifying the percentage of red, green, and blue; *(0, 0, 0)*=black, *(1, 1, 1)*=white. The commonly used RGB color values in the range 0–255 must be divided by 255 in order to scale them to the range 0–1 as required by PDFlib.
- *cmyk:* Four CMYK values between 0 = no color and 1 = full color, representing cyan, magenta, yellow, and black values; *(0, 0, 0, 0)*=white, *(0, 0, 0, 1)*=black. Note that this is different from the RGB specification.
- *iccbasedgra/rgb/cmyk:* ICC-based colors are specified with the help of an ICC profile.
- *spot:* Spot color (separation color space): a predefined or arbitrarily named custom color with an alternate representation in one of the other color spaces above; this is generally used for preparing documents which are intended to be printed on an offset printing machine with one or more custom colors. The tint value (percentage) ranges from 0 = no color to 1 = maximum intensity of the spot color.
- *lab:* Device-independent colors in the CIE L*a*b* color space are specified by a luminance value in the range 0-100 and two color values in the range -127 to 128.
- *pattern:* tiling pattern with an object composed of arbitrary text, vector, or image graphics.
- Shadings (smooth blends) provide a gradual transition between two colors, and are based on another color space. Shadings can be created with *PDF_shading( )*.
- The indexed color space is a not really a color space on its own, but rather an efficient coding of another color space. It will automatically be generated when an indexed (palette-based) image is imported, but cannot be specified directly.

The default color for stroke and fill operations is black.

Color specification in option lists. See Section 1.2, »Option Lists«, page 6, for a description and examples of the color data type in option lists.

C++ Java ***void setcolor(String fstype, String colorspace, double c1, double c2, double c3, double c4)***

Perl PHP ***PDF_setcolor(resource p, string fstype, string colorspace, float c1, float c2, float c3, float c4)***

C ***void PDF_setcolor(PDF *p,***
const char *fstype, const char *colorspace, double c1, double c2, double c3, double c4)

Set the current color space and color.

fstype One of *fill, stroke,* or *fillstroke* to specify that the color is set for filling, stroking, or both.

colorspace One of *gray, rgb, cmyk, spot, pattern, iccbasedgray, iccbasedrgb, iccbasedcmyk,* or *lab* to specify the color space.

PDF/X-1a: *colorspace=rgb*, *iccbasedgray/rgb/cmyk,* and *lab* are not allowed.

PDF/X-3: *colorspace=gray* requires that the *defaultgray* option in *PDF_begin_page_ext( )* has been set unless the PDF/X output intent is a grayscale or CMYK device. *colorspace= rgb* requires that the *defaultrgb* option in *PDF_begin_page_ext( )* has been set unless the PDF/X output intent is an RGB device. *colorspace=cmyk* requires that the *defaultcmyk* option in *PDF_begin_page_ext( )* has been set unless the PDF/X output intent is a CMYK device. Using *iccbasedgray/rgb/cmyk* and *lab* color requires an ICC profile in the output intent (a standard name is not sufficient in this case).

PDF/A: *colorspace=gray* requires that an output intent (any type) has been specified. *colorspace= rgb* or *cmyk* requires that an RGB or CMYK output intent has been specified, respectively.

c1, c2, c3, c4 Color components for the chosen color space. The interpretation of these values depends on the *colorspace* parameter:

- *gray: c1* specifies a gray value;
- *rgb: c1, c2, c3* specify red, green, and blue values.
- *cmyk: c1, c2, c3, c4* specify cyan, magenta, yellow, and black values;
- *iccbasedgray*: *c1* specifies a gray value;
- *iccbasedrgb: c1, c2, c3* specify red, green, and blue values;
- *iccbasedcmyk: c1, c2, c3, c4* specify cyan, magenta, yellow, and black values;
- *spot*: *c1* specifies a spot color handle returned by *PDF_makespotcolor( ),* and *c2* specifies a tint value between 0 and 1;
- *lab: c1, c2,* and *c3* specify color values in the CIE L*a*b* color space, interpreted with the D50 illuminant. *c1* specifies the L* (luminance) in the range 0 to 100, and *c2, c3* specify the a*, b* (chrominance) values in the range -127 to 128.
- *pattern*: *c1* specifies a pattern handle returned by *PDF_begin_pattern( )* or *PDF_ shading_pattern( ).*

Details All color values for the *gray, rgb,* and *cmyk* color spaces and the *tint* value for the *spot* color space must be numbers in the inclusive range 0–1. Unused parameters should be set to 0.

The fill and stroke color values for the *gray, rgb,* and *cmyk* color spaces are set to a default value of black at the beginning of each page. There are no defaults for spot and pattern colors.

If the *iccbasedgray/rgb/cmyk* color spaces are used, a suitable ICC profile must have been set before using the *setcolor:iccprofilegray/rgb/cmyk* parameters (see Table 6.3).

Scope *page, pattern* (only if the pattern's paint type is 1), *template, glyph* (only if the Type 3 font's *colorized* option is *true), document;* a pattern color can not be used within its own definition. Setting the color in *document* scope may be useful for defining spot colors with *PDF_makespotcolor( )*.

Params *setcolor:iccprofilegray/rgb/cmyk*

C++ Java ***int makespotcolor(String spotname)***

Perl PHP ***int PDF_makespotcolor(resource p, String spotname)***

C ***int PDF_makespotcolor(PDF *p, const char *spotname, int reserved)***

Find a built-in spot color name, or make a named spot color from the current fill color.

spotname The name of a built-in spot color, or an arbitrary name for the spot color to be defined. This name is restricted to a maximum length of 126 bytes. Only 8-bit characters are supported in the spot color name; Unicode or embedded null characters are not supported. PANTONE® colors are not supported in PDF/X-1a mode.

reserved (C language binding only) Reserved, must be 0.

Returns A color handle which can be used in subsequent calls to *PDF_setcolor( )* throughout the document. Spot color handles can be reused across all pages, but not across documents. There is no limit for the number of spot colors in a document.

Details If *spotname* is known in the internal color tables and the *spotcolorlookup* parameter is *true* (which is default), the supplied spot color name will be used. Otherwise the (CMYK or other) color values of the current fill color will be used to define the appearance of a new spot color. These alternate values will only be used for screen preview and low-end printing. The supplied spot color name will be used for producing color separations instead of the alternate values.

If *spotname* has already been used in a previous call to *PDF_makespotcolor( )*, the return value will be the same as in the earlier call, and will not reflect the current color.

The special spot color name *All* can be used to apply color to all color separations, which is useful for painting registration marks. A spot color name of *None* will produce no visible output on any color separation.

Scope *page, pattern, template, glyph* (only if the Type 3 font's *colorized* option is *true), document;* the current fill color must not be a spot color or pattern if a custom color is to be defined.

Params *spotcolorlookup, preserveoldpantonenames*

6.2 ICC Profiles

C++ Java ***int load_iccprofile(String profilename, String optlist)***

Perl PHP ***int PDF_load_iccprofile(resource p, string profilename, string optlist)***

C ***int PDF_load_iccprofile(PDF *p, const char *profilename, int len, const char *optlist)***

Search for an ICC profile and prepare it for later use.

profilename (Name string) The name of an *ICCProfile* resource, a disk-based or virtual file name, or one of the standard output condition names for PDF/X listed in Table 6.5. The latter is only allowed if the *usage* option is set to *outputintent*. Additional standard output intents can be defined withe the *StandardOutputIntent* resource.

len (C language binding only) Length of *profilename* (in bytes) for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

optlist An option list describing aspects of the profile handling according to Table 6.2. The following options can be used: *description, embedprofile, errorpolicy, metadata, usage*

Table 6.2 Options for PDF_load_iccprofile( )

key	explanation and possible values
description	*(String) This option is only used if* `usage=outputintent`. *It contains the human-readable description of the ICC profile which will be used along with the PDF/X output intent. Default: if* `profilename` *refers to a standard output intent, the description will be taken from an internal list; otherwise there will be no description.*
embedprofile	*(Boolean) This option is only used if* `usage=outputintent`. *Force an embedded ICC profile even if a standard output intent for PDF/X has been provided as* `profilename`. *Default:* `false`
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
metadata	*(Option list; PDF 1.4) Supply metadata for the profile (see Section 12.2, »XMP Metadata«, page 161)*
usage	*(Keyword) Describes the intended use of the ICC profile (default:* `iccbased`*)* ***iccbased*** *The ICC profile will be used to define an ICC-based color space for text or graphics, or will be applied to an image.* ***outputintent*** *The ICC profile will be used to define a PDF/X or PDF/A output intent.*

Returns A profile handle which can be used in subsequent calls to *PDF_load_image( )* or for setting profile-related parameters. The return value must be checked for -1 (in PHP: 0) which signals an error. The returned profile handle can not be reused across multiple PDF documents. Also, the returned handle can not be applied to an image if the *usage* option is *outputintent*. There is no limit to the number of ICC profiles in a document. If the function call fails you can request the reason of the failure with *PDF_get_errmsg( )*.

Details If the *usage* option is *iccbased* the named profile will be searched according to the profile search strategy. If the profile is found, it will be checked whether it is suitable (e.g. number of color components). The *sRGB* profile is always available internally, and must not be configured.

If the *usage* option is *outputintent* the named profile is first searched in an internal list of standard output intents. If this search was unsuccessful, the name will be

searched in the list of user-configured output intents. If the supplied name was found to be a standard output intent according to the built-in or user-configured list, no ICC profile will be searched, and the name supplied with the *description* option will be embedded in the PDF output as the output intent. If the name was not found to be a standard output intent identifier, it is treated as a profile name and the corresponding ICC profile will be embedded in the PDF as output intent.

PDF/X: the output intent must be set either using this function or by copying an imported document's output intent using *PDF_process_pdi( )*.

PDF/A: the output intent can be set using this function or by copying an imported document's output intent using *PDF_process_pdi( )*. However, if only device-independent colors are used in the document no output intent is required.

Scope *document;* the output intent should be set immediately after *PDF_begin_document( )*. If *usage=iccbased* the following scopes are also allowed: *page, pattern, template, glyph*.

Params See Table 6.3 and Table 6.4

Table 6.3 Keys for PDF_get/set_parameter( ) (see Section 2.1, »Parameter Handling«, page 11)

key	*explanation*
ICCProfile StandardOutputIntent	*The corresponding resource file line as it would appear for the respective category in a UPR file. Multiple calls add new entries to the internal list (see also resourcefile in Table 2.1). Scope: any*
iccwarning	*Deprecated, use* `errorpolicy`

Table 6.4 Keys for PDF_get/set_value( ) (see Section 2.1, »Parameter Handling«, page 11)

key	*explanation*
icccomponents	*Number of color components in the ICC profile referenced by the handle provided in the modifier*
setcolor:iccprofilegray	*ICC profile which specifies a Gray color space for use with PDF_setcolor( ). Scope: document, page, pattern, template, glyph*
setcolor:iccprofilergb	*ICC profile which specifies an RGB color space for use with PDF_setcolor( ). Scope: document, page, pattern, template, glyph*
setcolor:iccprofilecmyk	*ICC profile which specifies a CMYK color space for use with PDF_setcolor( ). Scope: document, page, pattern, template, glyph*

Table 6.5 Standard output intents for PDF/X (see `www.color.org` *for more information)*

Name	Printing process
CGATS TR 001	*SWOP (Publication) printing in USA: ANSI CGATS.6.*
FOGRA27	*ISO/DIS 12647-2:2004, Offset commercial and specialty printing according to ISO 12647-2, positive plates, paper type 1 or 2 (gloss or matte coated offset, 115 g/m2), screen frequency 60/cm.*
FOGRA28	*ISO/DIS 12647-2:2004, Offset commercial and specialty printing according to ISO 12647-2, positive plates, paper type 3 (coated web, 60 g/m2), screen frequency 60/cm.*
FOGRA29	*ISO/DIS 12647-2:2004, Offset commercial and specialty printing according to ISO 12647-2, positive plates, paper type 4 (uncoated white offset, 120 g/m2), screen frequency 60/cm.*
FOGRA30	*ISO/DIS 12647-2:2004, Offset commercial and specialty printing according to ISO 12647-2, positive plates, paper type 5 (uncoated, slightly yellowish, offset, 115 g/m2), screen frequency 60/cm.*
FOGRA31	*ISO/DIS 12647-2:2003, Continuous forms printing according to ISO 12647-2, positive plates, paper type 2 (matt coated offset, 115 g/m2), screen frequency 60/cm.*
FOGRA32	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, positive plates, paper type 4 (white uncoated offset, 80 g/m2), screen frequency 60/cm.*
FOGRA33	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, positive plates, paper type 2 (matte coated offset, 115 g/m2), screen frequency 54/cm.*
FOGRA34	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, positive plates, paper type 4 (white uncoated offset, 120 g/m2), screen frequency 60/cm.*
FOGRA35	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 2 (matte coated offset, 115 g/m2), screen frequency 54/cm.*
FOGRA36	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 4 (white uncoated offset, 120 g/m2), screen frequency 54/cm.*
FOGRA37	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 2 (matte coated offset, 115 g/m2), screen frequency 60/cm.*
FOGRA38	*ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 4 (white uncoated offset, 120 g/m2), screen frequency 60/cm.*
IFRA26	*ISO/DIS 12647-3:2004, Coldset offset printing, contact exposed negative acting plates or computer to plate (tone value increase of 26%), newsprint, screen ruling 40 lines per cm.*
IFRA30	*ISO/DIS 12647-3:2004, Coldset offset printing, contact exposed negative acting plates or computer to plate (tone value increase of 30%), newsprint, screen ruling 40 lines per cm. (Principally applicable to the USA).*
JC200103	*Japan Color 2001 Coated: ISO 12647-2:2004, sheet-fed offset printing, positive plates, paper type 3, (coated, 105 gsm), screen frequency 69/cm.*
JC200104	*Japan Color 2001 Uncoated: ISO 12647-2:2004, sheet-fed offset printing, positive plates, paper type 4, (uncoated, 105 gsm), screen frequency 69/cm.*
JCN2002	*Japan Color 2002 for Newspaper Printing: ISO/DIS 12647-3:2004, coldset offset printing, negative plates, newsprint, screen frequency 39/cm.*
JCW2003	*Japan Color 2003 for Web Offset: ISO 12647-2:2004, heat-set web offset printing, positive plates, paper type 3, (coated, 70 gsm), screen frequency 69/cm.*
The following standard output intents are deprecated, and should not be used although they are still available: FOGRA11-FOGRA26, IFRA 22, IFRA 28, OF COM PO P1 F60, OF COM PO P2 F60, OF COM PO-P3 F60, OF COM PO P4 F60, OF COM NE P1 F60, OF COM NE P2 F60, OF COM NE P3 F60, OF COM NE P4 F60, SC GC2 CO F30, Ifra_NP_40lcm_neg+CTP_05.00	

6.3 Patterns and Shadings

C++ Java ***int begin_pattern(double width, double height, double xstep, double ystep, int painttype)***

Perl PHP ***int PDF_begin_pattern(resource p, float width, float height, float xstep, float ystep, int painttype)***

C ***int PDF_begin_pattern(PDF *p,***
double width, double height, double xstep, double ystep, int painttype)

Start a pattern definition.

width, height The dimensions of the pattern's bounding box in points.

xstep, ystep The offsets when repeatedly placing the pattern to stroke or fill some object. Most applications will set these to the pattern *width* and *height*, respectively.

painttype If *painttype* is 1 the pattern must contain its own color specification which will be applied when the pattern is used; if *painttype* is 2 the pattern must not contain any color specification but instead the current fill or stroke color will be applied when the pattern is used for filling or stroking.

Returns A pattern handle that can be used in subsequent calls to *PDF_setcolor( )* during the enclosing *document* scope.

Details This function will reset all text, graphics, and color state parameters to their defaults, and establish a coordinate system according to the global *topdown* parameter. Hypertext functions and functions for opening images must not be used during a pattern definition, but all text, graphics, and color functions (with the exception of the pattern which is in the process of being defined) can be used.

Scope *document*, *page*; this function starts *pattern* scope, and must always be paired with a matching *PDF_end_pattern( )* call.

Params *topdown*

C++ Java ***void end_pattern( )***

Perl PHP ***PDF_end_pattern(resource p)***

C ***void PDF_end_pattern(PDF *p)***

Finish a pattern definition.

Scope *pattern;* this function terminates *pattern* scope, and must always be paired with a matching *PDF_begin_pattern( )* call.

C++ Java ***int shading_pattern(int shading, String optlist)***

Perl PHP ***int PDF_shading_pattern(resource p, int shading, string optlist)***

C ***int PDF_shading_pattern(PDF *p, int shading, const char *optlist)***

Define a shading pattern using a shading object (requires PDF 1.4).

shading A shading handle returned by *PDF_shading( )*.

optlist An option list describing aspects of the shading pattern according to Table 6.6. The following option can be used: *gstate*

Returns A pattern handle that can be used in subsequent calls to *PDF_setcolor( )* during the enclosing *document* scope.

Details This function can be used to fill arbitrary objects with a shading. To do so, a shading handle must be retrieved using *PDF_shading( ),* then a pattern must be defined based on this shading using *PDF_shading_pattern( ).* Finally, the pattern handle can be supplied to *PDF_setcolor( )* to set the current color to the shading pattern.

Scope *document, page, font*

Table 6.6 Options for PDF_shading_pattern( )

key	explanation and possible values
gstate	*(Handle) A graphics state handle*

C++ Java ***void shfill(int shading)***

Perl PHP ***PDF_shfill(resource p, int shading)***

C ***void PDF_shfill(PDF *p, int shading)***

Fill an area with a shading, based on a shading object (requires PDF 1.4).

shading A shading handle returned by *PDF_shading( ).*

Details This function allows shadings to be used without involving *PDF_shading_pattern( )* and *PDF_setcolor( ).* However, it works only for simple shapes where the geometry of the object to be filled is the same as that of the shading itself. Since the current clip area will be shaded (subject to the *extendo* and *extend1* options of the shading) this function will generally be used in combination with *PDF_clip( ).*

Scope *page, pattern* (only if the pattern's paint type is 1)*, template, glyph* (only if the Type 3 font's *colorized* option is *true), document*

C++ Java ***int shading(String shtype, double x0, double y0, double x1, double y1, double c1, double c2, double c3, double c4, String optlist)***

Perl PHP ***int PDF_shading(resource p, string shtype, float x0, float y0, float x1, float y1, float c1, float c2, float c3, float c4, string optlist)***

C ***int PDF_shading(PDF *p, const char *shtype, double x0, double y0, double x1, double y1, double c1, double c2, double c3, double c4, const char *optlist)***

Define a blend from the current fill color to another color (requires PDF 1.4 or above).

shtype The type of the shading; must be *axial* for linear shadings or *radial* for circle-like shadings.

x0, y0, x1, y1 For axial shadings, *(x0, y0)* and *(x1, y1)* are the coordinates of the starting and ending points of the shading. For radial shadings these points specify the centers of the starting and ending circles.

c1, c2, c3, c4 Color values of the shading's endpoint, interpreted in the current fill color space in the same way as the color parameters in *PDF_setcolor( ).* If the current fill color space is a spot color space *c1* will be ignored, and *c2* contains the tint value.

optlist An option list describing aspects of the shading according to Table 6.7. The following options can be used: *antialias, extendo, extend1, N, r0, r1*

Returns A shading handle that can be used in subsequent calls to *PDF_shading_pattern( )* and *PDF_shfill( )* during the enclosing *document* scope.

Details The current fill color will be used as the starting color; it must not be based on a pattern.

Scope *document, page, font*

Table 6.7 Options for PDF_shading( )

key	***explanation and possible values***
antialias	*(Boolean) Specifies whether to activate antialiasing for the shading. Default:* `false`
extendo	*(Boolean) Specifies whether to extend the shading beyond the starting point. Default:* `false`
extend1	*(Boolean) Specifies whether to extend the shading beyond the endpoint. Default:* `false`
N	*(Float) Exponent for the color transition function; must be > 0. Default: 1*
r0	*(Float; only for radial shadings, and required in this case) Radius of the starting circle*
r1	*(Float; only for radial shadings, and required in this case) Radius of the ending circle*

7 Image and Template Functions

Table 7.1 and Table 7.2 list relevant parameter and value key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 7.1 Image-related keys for PDF_get/set_parameter( )

key	explanation
honoriccprofile	*Read ICC color profiles embedded in images, and apply them to the image data. Default:* `true`
imagewarning	*Deprecated, use* `errorpolicy`
renderingintent	*The rendering intent for images. Default:* `Auto`. **Auto** *Do not specify any rendering intent in the PDF file, but use the device's default intent instead. Typical use: unknown cases* **AbsoluteColorimetric** *No correction for the device's white point (such as paper white) is made. Colors which are out of gamut are mapped to nearest value within the device's gamut. Typical use: exact reproduction of solid colors; not recommended for other uses* **RelativeColorimetric** *The color data is scaled into the device's gamut, mapping the white points onto one another while slightly shifting colors. Typical use: vector graphics* **Saturation** *Saturation of the colors will be preserved while the color values may be shifted. Typical use: business graphics* **Perceptual** *Color relationships are preserved by modifying both in-gamut and out-of-gamut colors in order to provide a pleasing appearance. Typical use: scanned images*

Table 7.2 Image-related keys for PDF_get/set_value( )

key	explanation
imagewidth[1] **imageheight**[1]	*Get the width or height, respectively, of an image in pixels. The modifier is the integer handle of the selected image. Scope: page, pattern, template, glyph, document, path*
image:iccprofile[1]	*Return a handle for the ICC profile embedded in the image referenced by the image handle provided in the modifier.*
orientation[1]	*Get the orientation value of an image. The modifier is the integer handle of the selected image. For TIFF images containing an orientation tag the value of this tag will be returned; in all other cases 1 will be returned. PDFlib will automatically compensate orientation values different from 1. Scope: page, pattern, template, glyph, document, path*
resx[1] **resy**[1]	*Get the horizontal or vertical resolution of an image, respectively. The modifier is the integer handle of the selected image. Scope: page, pattern, template, glyph, document, path.* *If the value is positive, the return value is the image resolution in pixels per inch (dpi). If the return value is negative,* `resx` *and* `resy` *can be used to find the aspect ratio of non-square pixels, but don't have any absolute meaning. If the return value is zero, the resolution of the image is unknown.*

1. Only for PDF_get_value( )

7.1 Images

C++ Java ***int load_image(String imagetype, String filename, String optlist)***

Perl PHP ***int PDF_load_image(resource p, string imagetype, string filename, string optlist)***

C ***int PDF_load_image(PDF *p,***
const char *imagetype, const char *filename, int len, const char *optlist)

Open a disk-based or virtual image file subject to various options.

imagetype The string *auto* instructs PDFlib to automatically detect the image file type (this is not possible for CCITT and raw images). Explicitly specifying the image format with one of the strings *bmp, ccitt, gif, jpeg, jpeg2000, png*, *raw*, or *tiff* offers slight performance advantages. Type *jpeg2000* requires PDF 1.5 or above, and is not allowed in PDF/A or PDF/X mode. Type *ccitt* is different from a TIFF file which contains CCITT-compressed image data.

filename (Name string) Generally the name of the image file to be opened. This must be the name of a disk-based or virtual file; PDFlib will not pull image data from URLs.

If a file with the specified file name cannot be found and *imagetype=auto* PDFlib will try to determine the appropriate file name suffix automatically; it will append all suffixes from the following list (in both lowercase and uppercase) to the specified *filename* and try to locate a file with that name in the directories specified in the searchpath:

.bmp, .ccitt, .g3, .g4, .fax, .gif, .jpg .jpeg, .jpx, .jp2, .jpf, .jpm, .j2k, .png, .raw, .tif, .tiff

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

optlist An option list specifying image-related properties according to Table 7.3. The following options can be used:

- Color-related options: *colorize, honoriccprofile, iccprofile, invert, renderingintent*
- Clipping and masking options: *clippingpathname, honorclippingpath, ignoremask, mask, masked*
- Special PDF features for using the image: *iconname, template*
- Options for raw and CCITT images: *bitreverse, bpc, components, height, K, width*
- Options for processing the image data: *ignoreorientation, inline, page, passthrough*
- Other options: *hypertextencoding, errorpolicy, interpolate, layer, metadata, OPI-1.3, OPI-2.0*

Returns An image handle which can be used in subsequent image-related calls. The return value must be checked for -1 (in PHP: 0) which signals an error. The returned image handle can not be reused across multiple PDF documents. If the function call fails you can request the reason of the failure with *PDF_get_errmsg( )*.

Details This function opens and analyzes a raster graphics file in one of the supported formats as determined by the *imagetype* parameter, and copies the relevant image data to the output document. This function will not have any visible effect on the output. In order to actually place the imported image somewhere in the generated output document, *PDF_fit_image( )* must be used. Opening the same image more than once per generated document is not recommended because the actual image data will be copied to the output document more than once.

PDFlib will open the image file with the provided *filename*, process the contents, and close the file before returning from this call. Although images can be placed multiply within a document (see *PDF_fit_image( )*), the actual image file will not be kept open after this call.

If *imagetype=raw* or *ccitt*, the *width, height, components,* and *bpc* options must be supplied since PDFlib cannot deduce those from the image data. The user is responsible for supplying option values which actually match the image. Otherwise corrupt PDF output may be generated, and Acrobat may respond with the message *Insufficient data for an Image*.

If *imagetype=raw*, the length of the supplied image data must be equal to *[width x components x bpc / 8] x height* bytes, with the bracketed term adjusted upwards to the next integer. The image samples are expected in the standard PostScript/PDF ordering, i.e. top to bottom and left to right (assuming no coordinate transformations have been applied). 16-bit samples must be provided with the most significant byte first (big-endian or »Mac« byte order). The polarity of the pixel values is as discussed in Section , »Color spaces«, page 101. If *bpc* is smaller than 8, each pixel row begins on a byte boundary, and color values must be packed from left to right within a byte. Image samples are always interleaved, i.e. all color values for the first pixel are supplied first, followed by all color values for the second pixel, and so on.

PDF/X-1a: RGB images are not allowed.

PDF/X-3: Grayscale images require that the *defaultgray* option in *PDF_begin_page_ext( )* must have been set unless the PDF/X output intent is a grayscale or CMYK device. RGB images require that the *defaultrgb* option in *PDF_begin_page_ext( )* must have been set unless the PDF/X output intent is an RGB device. CMYK images require that the *defaultcmyk* option in *PDF_begin_page_ext( )* must have been set unless the PDF/X output intent is a CMYK device.

PDF/A: Grayscale images require that an output intent has been specified. RGB or CMYK images require that an RGB or CMYK output intent has been specified, respectively.

Scope If the *inline* option is not provided, the scope is *document, page, font,* and this function must always be paired with a matching *PDF_close_image( )* call. Loading images in *document* or *font* scope instead of *page* scope offers slight output size advantages.
If the *inline* option is provided, the scope is *page, pattern, template, glyph,* and *PDF_close_image( )* must not be called.

Params See Table 7.1 and Table 7.2

Table 7.3 Options for PDF_load_image( )

key	explanation
bitreverse	*(Boolean; only for* `imagetype=ccitt`*) If* `true`*, do a bitwise reversal of all bytes in the compressed data. Default:* `false`
bpc	*(Integer; only for* `imagetype=raw`*; required in this case) Number of bits per component; must be 1, 2, 4, or 8. In PDF 1.5,* `bpc=16` *is also allowed.*
clipping-pathname	*(String; only for* `imagetype=tiff` *and* `jpeg`*; will be ignored if* `honorclippingpath=false`*) Read the path with the specified name from the image file and use it as clipping path. The named path must be present in the image file. Default: name of the path which is provided as clipping path in the image file*

Table 7.3 Options for PDF_load_image( )

key	explanation
colorize	*(Spot color handle; will be ignored if the* iccprofile *option is provided; not for* imagetype=jpeg2000*) Colorize the image with a spot color handle, which must have been retrieved with PDF_makespotcolor( ). The image must be a black and white or grayscale image.*
components	*(Integer; only for* imagetype=raw*; required in this case) Number of image components (channels); must be 1, 3, or 4.*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
height	*(Integer; only for* imagetype=raw *and* ccitt*; required in this case) Image height in pixels.*
honoriccprofile	*(Boolean; only for* imagetype=jpeg, png*, and* tiff*; will be set to false if the* colorize *option is provided) Read an embedded ICC profile (if any) and apply it to the image. Default: the value of the* honoriccprofile *parameter.*
honorclippingpath	*(Boolean; only for* imagetype=tiff *and* jpeg*) Read the clipping path from the image file if available, and apply it to the image. Default:* true
hypertextencoding	*(Keyword) Specifies the encoding for the* iconname *option. An empty string is equivalent to* unicode*. Default: value of the global* hypertextencoding *parameter.*
iccprofile	*(ICC handle; only for* imagetype=jpeg, png*, and* tiff*) Handle of an ICC profile which will be applied to the image. Default: an embedded profile if one is present in the image and* honoriccprofile=true*.*
iconname	*(Hypertext string; will be ignored if* inline=true*; forces* template=true*) Attaches a name to the image so that it can be referenced via JavaScript, e.g. to use the image as an icon for form fields.*
ignoremask	*(Boolean) Ignores transparency information in the image. Default: false*
ignoreorientation	*(Boolean; only for* imagetype=tiff*) Ignores any orientation tag in the image. This may be useful for compensating wrong orientation information. Default:* false
imagewarning	*Deprecated, use* errorpolicy
inline	*(Boolean; only for* imagetype=ccitt, jpeg*, and* raw*; only recommended when loading bitmap glyphs for Type 3 fonts) If true, the image will be written directly into the content stream of the page, pattern, template, or glyph description (only recommended for the glyphs of Type 3 fonts).*
interpolate	*(Boolean; must be* false *for PDF/A) Enables image interpolation to improve the appearance on screen and paper. This is useful for bitmap images for glyph descriptions in Type 3 fonts. Default:* false
invert	*(Boolean; not for* imagetype=jpeg2000 *unless* mask=true*) Inverts the image (swap light and dark colors). This can be used as a workaround for images which are interpreted differently by applications. Default:* false
K	*(Integer; only for* imagetype=ccitt*) CCITT parameter for compression scheme selection. Default: 0* **-1** *G4 compression* **0** *One-dimensional G3 compression (G3-1D)* **1** *Mixed one- and two-dimensional compression (G3, 2-D)*
layer	*(Layer handle; PDF 1.5) Layer to which the image will belong. The image will only be visible if the corresponding layer is visible. Note that the image will be tied to the layer. Use PDF_begin_layer( ) before placing the image if you need different placements of the same image to belong to different layers.*
mask	*(Boolean; only for images with one color component, including indexed color). The image is going to be used as a mask. This is required for 1-bit masks, but optional for masks with more than 1 bit per pixel. However, masks with more than 1 bit require PDF 1.4. Default:* false*. There are two uses for masks:* ► *Masking another image: The returned image handle may be used in subsequent calls for opening another image and can be supplied for the* masked *option.* ► *Placing a colorized transparent image: Treat the 0-bit pixels in the image as transparent, and colorize the 1-bit pixels with the current fill color.*

Table 7.3 Options for PDF_load_image( )

key	explanation
masked	*(Image handle) Image handle for an image which will be applied as a mask to the current image. The image handle has been returned by a previous call to PDF_load_image( ) and has not yet been closed. In PDF 1.3 compatibility mode the mask handle must refer to a 1-bit image and must have been loaded with the* mask *option. In PDF/A and PDF/X mode this option is only allowed with 1-bit masks.*
metadata	*(Option list; PDF 1.4) Supply metadata for the image (see Section 12.2, »XMP Metadata«, page 161).*
OPI-1.3	*(Option list; not for PDF/A and PDF/X) An option list containing OPI 1.3 PostScript comments as option names; the following entries are required:* ALDImageFilename *(string),* ALDImageDimensions *(list of integers),* ALDImageCropRect *(rectangle with integers),* ALDImagePosition *(list of floats)* *The following entries are optional:* ALDImageID *(string),* ALDObjectComments *(string),* ALDImageCropFixed *(rectangle),* ALDImageResolution *(list of floats),* ALDImageColorType *(keyword; one of* Process, Spot, Separation*; default:* Spot*),* ALDImageColorType *(list of four color values in the range 0...1 and a color name),* ALDImageTint *(float),* ALDImageOverprint *(boolean),* ALDImageType *(list of integers),* ALDImageGrayMap *(list of integers),* ALDImageTransparency *(boolean),* ALDImageAsciiTag *(list of integer/string pairs)* *The suboption* normalizefilename *controls the handling of file names: if* true*, file names will be normalized as mandated by the PDF reference. If* false *they will be copied to the output without any modification. The latter can be useful to deal with some OPI servers which do not properly process normalized file names. Default:* false
OPI-2.0	*(Option list; not for PDF/A and PDF/X) An option list containing OPI 2.0 PostScript comments as option names; the following entry is required:* ImageFilename *(string)* *The following entries should either both be present or absent:* ImageCropRect *(rectangle),* ImageDimensions *(list of floats)* *The following entries are optional:* MainImage *(string),* TIFFASCIITag *(list of integer/string pairs),* ImageOverprint *(boolean),* ImageInks *(the string* full_color*, the string* registration*, or a list containing the string* monochrome *and string/float pairs for each colorant name and tint),* IncludedImageDimensions *(list of integers),* IncludedImageQuality *(integer with one of the values 1, 2, or 3)* *The option* normalizefilename *is also supported (see OPI-1.3).*
page	*(Integer; only for* imagetype=gif *or* tiff*; must be 1 if used with other formats) Extract the image with the given number from a multi-page image file. The first image has the number 1. Default: 1*
passthrough	*(Boolean; only for* imagetype=tiff *or* jpeg*) Controls handling of TIFF and JPEG image data.* ***tiff*** *(Default:* true*) If* true*, compressed TIFF image data will be directly passed through to the PDF output if possible. Setting this option to* false *may help in cases where a TIFF image contains damaged or incomplete data.* ***jpeg*** *(Default:* false*) If* false*, PDFlib will transcode JPEG image data in order to clean up the data for compatibility with Acrobat. If* true*, JPEG image data will be directly copied to the PDF output. This option will be ignored for multiscan and certain CMYK JPEG images. Setting this option to* true *may speed up processing, but certain rare JPEG flavors won't display correctly in Acrobat.*
rendering-intent	*(Keyword) Rendering intent for the image. See Table 7.1 for a list of possible keywords and their meaning. Default: the value of the global* renderingintent *parameter*
template	*(Boolean) If* true*, generate a PDF Image XObject embedded in a Form XObject (called template in PDFlib) instead of a plain Image XObject. This can be useful for creating templates for form field icons which consist of an image only. It is also required for compatibility with certain OPI servers when using one of the* OPI-1.3 *or* OPI-2.0 *options. Default:* false*. Scope: document*
width	*(Integer; only for* imagetype=raw *and* ccitt*; required in this case) Image width in pixels*

C++ Java ***void close_image(int image)***

Perl PHP ***PDF_close_image(resource p, int image)***

C ***void PDF_close_image(PDF *p, int image)***

Close an image.

image A valid image handle retrieved with *PDF_load_image( )*.

Details This function only affects PDFlib's associated internal image structure. If the image has been opened from file, the actual image file is not affected by this call since it has already been closed at the end of the corresponding *PDF_load_image( )* call. An image handle cannot be used any more after it has been closed with this function, since it breaks PDFlib's internal association with the image.

Scope *document, page, font;* must always be paired with a matching call to *PDF_load_image( )* unless the *inline* option has been used.

C++ Java ***void fit_image(int image, double x, double y, String optlist)***

Perl PHP ***PDF_fit_image(resource p, int image, float x, float y, string optlist)***

C ***void PDF_fit_image(PDF *p, int image, double x, double y, const char *optlist)***

Place an image or template at position *(x, y)* on the page, subject to various options.

image A valid image or template handle retrieved with one of the *PDF_load_image( )* or *PDF_begin_template_ext( )* functions.

x, y The coordinates of the reference point in the user coordinate system where the image or template will be located, subject to various options.

optlist An option list specifying placement details according to Table 7.4. The following options can be used:
adjustpage, blind, boxsize, dpi, fitmethod, ignoreclippingpath, ignoreorientation, matchbox, orientate, position, rotate, scale, showborder

Details The image or template (collectively referred to as an object below) will be placed relative to the reference point *(x, y)*. By default, the lower left corner of the object will be placed at the reference point. However, the *orientate, boxsize, position,* and *fitmethod* options can modify this behavior. By default, an image will be scaled according to its resolution value(s). This behavior can be modified with the *dpi, scale,* and *fitmethod* options.

Scope *page, pattern, template, glyph* (only if the Type 3 font's *colorized* option is true, or if the image is a mask)*;* this function can be called an arbitrary number of times on arbitrary pages, as long as the image handle has not been closed with *PDF_close_image( )*.

Table 7.4 Options for PDF_fit_image( ) and PDF_fit_pdi_page( )

key	explanation
adjustpage	*(Boolean; ignored if* `blind=true`*) Adjust the dimensions of the current page to the object such that the upper right corner of the page coincides with the upper right corner of the object plus (x, y) with the function parameters* x *and* y*. The MediaBox will be adjusted, and all other box entries will be reset to their defaults. With the value* 0 *for the* `position` *option the following useful cases shall be noted:* *x >= 0 and y >= 0* *The object is surrounded by a white margin. This margin has thickness y in horizontal direction and thickness x in vertical direction.* *x < 0 and y < 0* *Horizontal and vertical strips will be cropped from the image.* *This option is only effective in scope* `page`*, and must not be used when the* `topdown` *parameter has been set to* `true`*. Default:* `false`
blind	*(Boolean) If* `true`*, all positioning and scaling calculations will be done, but the object will not be placed on the output page. This is useful for processing the blocks on a page without actually using the page's contents. Default:* `false`
boxsize	*(List of floats) Two values specifying the width and height of a box, relative to which the object will be placed and possibly scaled. The lower left corner of the box coincides with the reference point (x, y). Placing the image and fitting it into the box is controlled by the* `position` *and* `fitmethod` *options. If width=0, only the height is considered; If height=0, only the width is considered. In these cases the object will be placed relative to the vertical line from* (`x`, `y`) *to* (`x`, `y+height`)*, or the horizontal line from* (`x`, `y`) *to* (`x+width`, `y`)*, respectively. Default:* {`0 0`}
dpi	*(List of floats) One or two values specifying the desired image resolution in pixels per inch in horizontal and vertical direction. If a single value is supplied it will be used for both dimensions. With the value* 0 *the image's internal resolution will be used if available, or 72 dpi otherwise. As an alternative to the value 0, the keyword* `internal` *can be supplied. The scaling resulting from this option is relative to the current user coordinate system; if it has been scaled the resulting physical resolution will be different from the supplied values.* *This option will be ignored for templates and PDF pages, or if the* `fitmethod` *option has been supplied with one of the keywords* `auto`, `meet`, `slice`, *or* `entire`*. Default:* `internal`
fitmethod	*(Keyword; will be ignored unless* `boxsize` *is supplied) Specifies the method used to fit the object into the box. Default:* `nofit` ***nofit*** *Position the object only, without any scaling or clipping.* ***clip*** *Position the object, and clip it at the edges of the box.* ***meet*** *Position the object according to the* `position` *option, and scale it so that it entirely fits into the box while preserving its aspect ratio. Generally at least two edges of the object will meet the corresponding edges of the box. The* `dpi` *and* `scale` *options are ignored.* ***auto*** *Same as* `meet`*.* ***slice*** *Position the object according to the* `position` *option, and scale it such that it entirely covers the box, while preserving the aspect ratio and making sure that at least one dimension of the object is fully contained in the box. Generally parts of the object's other dimension will extend beyond the box, and will therefore be clipped. The* `dpi` *and* `scale` *options are ignored.* ***entire*** *Position the object according to the* `position` *option, and scale it such that it entirely covers the box. Generally this method will distort the object. The* `dpi` *and* `scale` *options are ignored.*
ignore-clippingpath	*(Boolean; only for TIFF and JPEG images) A clipping path which may be present in the image file will be ignored. Default:* `false`*, i.e. the clipping path will be applied*
ignore-orientation	*(Boolean; only for TIFF images) Ignore any orientation tag in the image. This may be useful for compensating wrong orientation information. Default: the value of the* `ignoreorientation` *option in PDF_load_image( )*
matchbox	*(Option list) Option list with matchbox details according to Table 4.12.*

Table 7.4 Options for PDF_fit_image( ) and PDF_fit_pdi_page( )

key	*explanation*
orientate	*(Keyword) Specifies the desired orientation of the object when it is placed. Default:* `north` ***north*** *upright* ***east*** *pointing to the right* ***south*** *upside down* ***west*** *pointing to the left*
position	*(List of floats or keywords) One or two values specifying the position of the reference point* (`x`, `y`) *within the object with* {`0 0`} *being the lower left corner of the object, and* {`100 100`} *the upper right corner. If the* `boxsize` *option has been specified, the* `position` *option also specifies the positioning of the box, i.e. the corresponding point in the box will be placed at the reference point* (`x`, `y`). *The values are expressed as percentages of the object's width and height. If both percentages are equal it is sufficient to specify a single float value. Examples:* *0 or {0 0}* *lower left corner* *{50 100}* *middle of the top edge* *50 or {50 50}* *center of the object* *The keywords* `left`, `center`, `right` *(in x direction) or* `bottom`, `center`, `top` *(in y direction) can be used as equivalents for the values 0, 50, and 100. If only one keyword has been specified the corresponding keyword for the other direction will be added.* *Default:* {`left bottom`}
rotate	*(Float) Rotates the coordinate system, using the reference point as center and the specified value as rotation angle in degrees. This results in the box and the object being rotated. The rotation will be reset when the object has been placed. Default: 0*
scale	*(List of floats) Scales the object in horizontal and vertical direction by the specified scaling factors (not percentages). If both factors are equal it is sufficient to specify a single float value. This option will be ignored if the* `fitmethod` *option has been supplied with one of the keywords* `auto`, `meet`, `slice`, *or* `entire`. *Default: 1*
showborder	*(Boolean) If* `true`, *the border of the fitbox will be stroked (using the current graphics state). This may be useful for development and debugging. Default:* `false`

7.2 Templates

Note *The template functions described in this section are unrelated to variable data processing with PDFlib blocks. Use PDF_fill_textblock( ), PDF_fill_imageblock( ), and PDF_fill_pdfblock( ) to fill blocks prepared with the PDFlib block plugin (see Chapter 9, »Personalization Functions (PPS)«, page 133).*

C++ Java ***int begin_template_ext(double width, double height, String optlist)***

Perl PHP ***int PDF_begin_template_ext(resource p, float width, float height, string optlist)***

C ***int PDF_begin_template_ext(PDF *p, double width, double height, const char *optlist)***

Start a template definition.

width, height The dimensions of the template's bounding box in points.

optlist Option list specifying template-related properties.
The following options of *PDF_load_image( )* can be used (see Table 7.3):
iconname, layer, metadata, OPI-1.3, OPI-2.0
The following option of *PDF_begin_page_ext( )* can be used (see Table 2.9):
topdown

Returns A template handle which can be used in subsequent image-related calls, especially *PDF_fit_image( )*. There is no error return.

Details This function will reset all text, graphics, and color state parameters to their defaults, and establish a coordinate system according to the global *topdown* parameter. Hypertext functions and functions for opening images must not be used during a template definition, but all text, graphics, and color functions can be used.

Scope *document, page;* this function starts *template* scope, and must always be paired with a matching *PDF_end_template( )* call.

C++ Java ***void end_template( )***

Perl PHP ***PDF_end_template(resource p)***

C ***void PDF_end_template(PDF *p)***

Finish a template definition.

Scope *template;* this function terminates *template* scope, and must always be paired with a matching *PDF_begin_template( )* call.

C++ Java ***int begin_template(double width, double height)***

Perl PHP ***int PDF_begin_template(resource p, float width, float height)***

C ***int PDF_begin_template(PDF *p, double width, double height)***

Deprecated, use *PDF_begin_template_ext( )*.

7.3 Thumbnails

C++ Java ***void add_thumbnail(int image)***
Perl PHP ***PDF_add_thumbnail(resource p, int image)***
C ***void PDF_add_thumbnail(PDF *p, int image)***

Add an existing image as thumbnail for the current page.

image A valid image handle retrieved with *PDF_load_image( )*.

Details This function adds the supplied image as thumbnail image for the current page. A thumbnail image must adhere to the following restrictions:

- The image must be no larger than 106 x 106 pixels.
- The image must use the grayscale, RGB, or indexed RGB color space.
- Multi-strip TIFF images can not be used as thumbnails because thumbnails must be constructed from a single PDF image object.

This function doesn't generate thumbnail images for pages, but only offers a hook for adding existing images as thumbnails. The actual thumbnail images must be generated by the client. The client must ensure that color, height/width ratio, and actual contents of a thumbnail match the corresponding page contents.

Since Acrobat 5 and above generates thumbnails on the fly (though not Acrobat 5 or Adobe Reader 6 in the Browser), and thumbnails increase the overall file size of the generated PDF, it is recommended not to add thumbnails, but rely on client-side thumbnail generation instead.

Scope *page;* must only be called once per page. Not all pages need to have thumbnails attached to them.

8 PDF Import Functions (PDI)

Note *All functions described in this chapter require the additional PDF import library (PDI) which requires PDFlib+PDI or PDFlib Personalization Server (PPS), but is not part of PDFlib Lite and PDFlib. Please visit our Web site for more information on obtaining PDI.*

8.1 Document and Page

Table 8.1 lists relevant parameter key names for this section (see Section 2.1, »Parameter Handling«, page 11).

Table 8.1 PDI-related keys for PDF_get/set_parameter( )

key	explanation
pdi[1]	*Returns the string* `true` *if the PDI library is attached (which is not true for PDFlib and PDFlib Lite), and* `false` *otherwise. Scope: any, null*[2]
pdiwarning	*Deprecated, use* `errorpolicy`

1. Only for PDF_get_parameter( )
*2. May be called with a PDF * argument of NULL or 0*

C++ Java ***int open_pdi_document(String filename, String optlist)***
Perl PHP ***int PDF_open_pdi_document(resource p, string filename, string optlist)***
C ***int PDF_open_pdi_document(PDF *p, const char *filename, int len, const char *optlist)***

Open a disk-based or virtual PDF document and prepare it for later use.

filename (Name string) The name of the PDF file.

optlist An option list specifying PDF open options according to Table 8.2. The following options can be used: *infomode, inmemory, errorpolicy, password, repair, requiredmode*

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

Returns A PDI document handle which can be used for processing individual pages of the document or for querying document properties. A return value of -1 (in PHP: 0) indicates that the PDF document couldn't be opened. An arbitrary number of PDF documents can be opened simultaneously. The return value can be used until the end of the enclosing document scope. If the function call fails you can request the reason of the failure with *PDF_get_errmsg( )*.

Details By default, the document will be rejected if at least one of the following conditions is true:

- The document is damaged.
- The document uses a higher PDF version than the current PDF document.
- The document is encrypted, but the corresponding password has not been supplied in the *password* option.
- The document is not compatible to the current PDF/X or PDF/A output conformance level, or uses an incompatible output intent.

- The document is Tagged PDF, and the *tagged* option in *PDF_begin_document( )* is *true*.

Except for the first reason, the *infomode* option can be used to open the document nevertheless. This may be useful to query information about the PDF using the *PDF_pcos_get_*( )* functions, such as encryption, PDF/A or PDF/X status, document info fields, etc.

In order to get more detailed information about the nature of a PDF import-related problem (wrong PDF file name, unsupported format, bad PDF data, etc.), use *PDF_get_errmsg( )* to receive a more detailed error message.

PDF/A: the imported document must be compatible to the current PDF/A output conformance level and output intent unless *infomode=true*.

PDF/X: the imported document must be compatible to the current PDF/X output conformance level unless *infomode=true*, and must use the same output intent as the generated document.

Scope *object*, *document*, *page;* in *object* scope a PDI document handle can only be used in the *PDF_pcos_get_*( )* functions.

Table 8.2 Options for PDF_open_pdi_document( )

key	*explanation*
infomode	*(Boolean) If* `true`*, the document will be opened such that information can be queried with the pCOS interface, but the pages can not be imported into the current output document. In particular, the following kinds of documents can be opened when* `infomode=true` *(default:* `false` *if* `requiredmode=full`*, otherwise* `true`*):* - *PDFs which are not compatible to the current PDF/X or PDF/A conformance level* - *PDFs with a higher PDF version than the current document* - *Encrypted PDFs where the password is not known (exception: PDF 1.6 documents created with the Distiller setting »Object Level Compression: Maximum«)* - *Tagged PDF when the* `tagged` *option in PDF_begin_document( ) is* `true`
inmemory	*(Boolean) If* `true`*, PDI will load the complete file into memory and process it from there. This can result in a tremendous performance gain on some systems (especially MVS) at the expense of memory usage. If* `false`*, individual parts of the document will be read from disk as needed. Default:* `false`
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
password	*(String with a maximum length of 32 characters) Master password required to open a protected PDF document for import. If* `infomode=true` *the user password (which may even be empty) is sufficient to query document information. If no password has been supplied at all for an encrypted document the document handle can only be used to query its encryption status.*
pdiwarning	*Deprecated, use* `errorpolicy`
repair	*(Keyword) Specifies how to treat damaged PDF input documents. Repairing a document takes more time than normal parsing, but may allow processing of certain damaged PDFs. Note that some documents may be damaged beyond repair (default:* `auto`*):* ***force*** *Unconditionally try to repair the document, regardless of whether or not it has problems.* ***auto*** *Repair the document only if problems are detected while opening the PDF.* ***none*** *No attempt will be made at repairing the document. If there are problems in the PDF the function call will fail.*
requiredmode	*(Keyword) The minimum pcos mode* (`minimum`/`restricted`/`full`) *which is acceptable when opening the document. The call will fail if the resulting pcos mode would be lower than the required mode. If the call succeeds it is guaranteed that the resulting pcos mode is at least the one specified in this option. However, it may be higher; e.g.* `requiredmode=minimum` *for an unencrypted document will result in* `full` *mode. Default:* `full`

C++ Java ***int open_pdi(String filename, String optlist, int len)***

Perl PHP ***int PDF_open_pdi(resource p, string filename, string optlist, int len)***

C ***int PDF_open_pdi(PDF *p, const char *filename, const char *optlist, int len)***

Deprecated; use *PDF_open_pdi_document( )*.

C ***int PDF_open_pdi_callback(PDF *p, void *opaque, size_t filesize,***
size_t (*readproc)(void *opaque, void *buffer, size_t size),
int (*seekproc)(void *opaque, long offset), const char *optlist)

Open a PDF document from a custom data source and prepare it for later use.

opaque A pointer to some user data that might be associated with the input PDF document. This pointer will be passed as the first parameter of the callback functions, and can be used in any way. PDI will not use the opaque pointer in any other way.

filesize The size of the complete PDF document in bytes.

readproc A callback function which copies *size* bytes to the memory pointed to by *buffer*. If the end of the document is reached it may copy less data than requested. The function must return the number of bytes copied.

seekproc A callback function which sets the current read position in the document. *offset* denotes the position from the beginning of the document (0 meaning the first byte). If successful, this function must return 0, otherwise -1.

optlist An option list specifying PDF open options according to Table 8.2. The following options can be used:
infomode, inmemory, password, pdiwarning, requiredmode

Returns A PDI document handle which can be used for processing individual pages of the document or for querying document properties. A return value of -1 indicates that the PDF document couldn't be opened. An arbitrary number of PDF documents can be opened simultaneously. The return value can be used until the end of the enclosing document scope. If the function call fails you can request the reason of the failure with *PDF_get_errmsg( )*.

Details This is a specialized interface for applications which retrieve arbitrary chunks of PDF data from some data source instead of providing the PDF document in a disk file or in memory.

Scope *object, document, page;* in *object* scope a PDI document handle can only be used to query information from a PDF document.

Bindings Only available in the C binding.

C++ Java ***void close_pdi_document(int doc)***

Perl PHP ***PDF_close_pdi_document(resource p, int doc)***

C ***void PDF_close_pdi_document(PDF *p, int doc)***

Close all open PDI page handles, and close the input PDF document.

doc A valid PDF document handle retrieved with *PDF_open_pdi_document( )*.

Details This function closes a PDF import document, and releases all resources related to the document. All document pages which may be open are implicitly closed. The document handle must not be used after this call. A PDF document should not be closed if more pages are to be imported. Although you can open and close a PDF import document an arbitrary number of times, doing so may result in unnecessary large PDF output files.

Scope *object, document, page*

C++ Java ***void close_pdi(int doc)***

Perl PHP ***PDF_close_pdi(resource p, int doc)***

C ***void PDF_close_pdi(PDF *p, int doc)***

Deprecated; use *PDF_close_pdi_document( )*.

C++ Java ***int open_pdi_page(int doc, int pagenumber, String optlist)***

Perl PHP ***int PDF_open_pdi_page(resource p, int doc, int pagenumber, string optlist)***

C ***int PDF_open_pdi_page(PDF *p, int doc, int pagenumber, const char* optlist)***

Prepare a page for later use with *PDF_fit_pdi_page( )*.

doc A valid PDF document handle retrieved with *PDF_open_pdi_document( )*.

pagenumber The number of the page to be opened. The first page has page number 1.

optlist An option list specifying page options according to Table 8.3. The following options can be used: *errorpolicy, hypertextencoding, iconname, infomode, metadata, pdiusebox,*

Returns A page handle which can be used for placing pages with *PDF_fit_pdi_page( )*. A return value of -1 (in PHP: 0) indicates that the page couldn't be opened. The return value can be used until the end of the enclosing document scope. If the *infomode* option was *true* when the document has been opened with *PDF_open_pdi_document( ),* the handle can not be used with *PDF_fit_pdi_page( )*. If the function call fails you can request the reason of the failure with *PDF_get_errmsg( )*.

Details This function will copy all data comprising the imported page to the output document, but will not have any visible effect on the output. In order to actually place the imported page somewhere in the generated output document, *PDF_fit_pdi_page( )* must be used. In order to get more detailed information about a problem related to PDF import (unsupported format, bad PDF data, etc.) you can call *PDF_get_errmsg( )*.

This function will fail if the PDF version number of the imported document is higher than the PDF version number of the generated PDF output document.

An arbitrary number of pages can be opened simultaneously. If the same page is opened multiply, different handles will be returned, and each handle must be closed exactly once.

Scope *document, page*

Table 8.3 Options for PDF_open_pdi_page( )

key	explanation
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
hypertext-encoding	*(Keyword) Specifies the encoding for the* `iconname` *option. An empty string is equivalent to* `unicode`*. Default: value of the global* `hypertextencoding` *parameter*
iconname	*(Hypertext string) Attach a name to the imported page so that it can be referenced via JavaScript, e.g. to use the page as an icon for form fields.*
infomode	*Deprecated; use pCOS to query page properties without actually placing the page*
layer	*(Layer handle; PDF 1.5) Layer to which the page will belong. The page will only be visible if the corresponding layer is visible. Note that the page will be tied to the layer. Use PDF_begin_layer( ) before placing the page if you need different placements of the same page to belong to different layers.*
metadata	*(Option list; PDF 1.4) Supply metadata for the imported page (see Section 12.2, »XMP Metadata«, page 161)*
pdiusebox	*(Keyword) Specifies which box dimensions will be used for determining an imported page's size. Default:* `crop`*.* **media** *Use the MediaBox (which is always present)* **crop** *Use the CropBox if present, else the MediaBox* **bleed** *Use the BleedBox if present, else the CropBox* **trim** *Use the TrimBox if present, else the CropBox* **art** *Use the ArtBox if present, else the CropBox*
pdiwarning	*Deprecated, use* `errorpolicy`

C++ Java ***void close_pdi_page(int page)***
Perl PHP ***PDF_close_pdi_page(resource p, int page)***
C ***void PDF_close_pdi_page(PDF *p, int page)***

Close the page handle and free all page-related resources.

page A valid PDF page handle (not a page number!) retrieved with *PDF_open_pdi_page( )*.

Details This function closes the page associated with the page handle identified by *page*, and releases all related resources. *page* must not be used after this call.

Scope *document, page*

C++ Java ***void fit_pdi_page(int page, double x, double y, String optlist)***
Perl PHP ***PDF_fit_pdi_page(resource p, int page, float x, float y, string optlist)***
C ***void PDF_fit_pdi_page(PDF *p, int page, double x, double y, const char *optlist)***

Place an imported PDF page on the page subject to various options.

page A valid PDF page handle (not a page number!) retrieved with *PDF_open_pdi_page( )*. The *infomode* option must have been *false* when opening the document. The page handle must not have been closed.

x, y The coordinates of the reference point in the user coordinate system where the page will be located, subject to various options.

optlist An option list specifying scaling and placement details according to Table 7.4. The following options can be used:
adjustpage, blind, boxsize, dpi, fitmethod, ignoreclippingpath, ignoreorientation, matchbox, orientate, position, rotate, scale, showborder

Details This function is similar to *PDF_fit_image( ),* but operates on imported PDF pages instead.

Scope *page, pattern, template, glyph*

8.2 pCOS Functions

All pCOS functions work with paths designating the target object in the PDF document. pCOS paths are discussed in detail in the *PDFlib Tutorial*.

Note *In evaluation mode pCOS will accept input documents up to a maximum of 1 MB or 10 pages. However, the following elements can also be queried for larger documents in evaluation mode: page count, page dimensions, block details, and all universal pseudo objects.*

C++ Java ***double pcos_get_number(int doc, string path)***

Perl PHP ***double PDF_pcos_get_number(resource tet, long doc, string path)***

C ***double PDF_pcos_get_number(PDF *p, int doc, const char *path, ...)***

Get the value of a pCOS path with type *number* or *boolean*.

doc A valid document handle obtained with *PDF_open_pdi_document( )*.

path A full pCOS path for a numerical or boolean object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders *(%s* for strings or *%d* for integers; use *%%* for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns The numerical value of the object identified by the pCOS path. For Boolean values 1 will be returned if they are *true,* and 0 otherwise.

Scope any

C++ Java ***const string pcos_get_string(int doc, string path)***

Perl PHP ***string PDF_pcos_get_string(resource tet, long doc, string path)***

C ***const char *PDF_pcos_get_string(TET *tet, int doc, const char *path, ...)***

Get the value of a pCOS path with type *name, string* or *boolean*.

doc A valid document handle obtained with *PDF_open_pdi_document( )*.

path A full pCOS path for a name, string, or boolean object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders *(%s* for strings or *%d* for integers; use *%%* for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns A string with the value of the object identified by the pCOS path. For Boolean values the strings *true* or *false* will be returned.

Details This function will raise an exception if pCOS does not run in full mode and the type of the object is *string*. As an exception, the objects */Info/** (document info keys) can also be

retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*, and *bookmarks[...]/Title* and *annots[...]/contents* can be retrieved in restricted pCOS mode if *nocopy=false*.

This function assumes that strings retrieved from the PDF document are text strings. String objects which contain binary data should be retrieved with *PDF_pcos_get_stream( )* instead which does not modify the data in any way.

Scope any

Bindings C and C++ language bindings: The string will be returned in UTF-8 format.
C binding: The returned string can be used until the next call to this function.

C++ Java ***const unsigned char *pcos_get_stream(int doc, int *length, string optlist, string path)***

Perl PHP ***string PDF_pcos_get_stream(resource tet, long doc, string path)***

C ***const unsigned char *PDF_pcos_get_stream(TET *tet, int doc, int *length, const char *optlist, const char *path, ...)***

Get the contents of a pCOS path with type *stream*, *fstream*, or *string*.

doc A valid document handle obtained with *PDF_open_pdi_document( )*.

length (C and C++ language bindings only) A pointer to a variable which will receive the length of the returned stream data in bytes.

optlist Reserved, must be an empty string.

path A full pCOS path for a stream or string object.

Additional parameters (C language binding only) A variable number of additional parameters can be supplied if the *key* parameter contains corresponding placeholders *(%s* for strings or *%d* for integers; use *%%* for a single percent sign). Using these parameters will save you from explicitly formatting complex paths containing variable numerical or string values. The client is responsible for making sure that the number and type of the placeholders matches the supplied additional parameters.

Returns The unencrypted data contained in the stream or string. The returned data will be empty (in C and C++: NULL) if the stream is empty.

If the object has type *stream,* all filters will be removed from the stream contents (i.e. the actual raw data will be returned). If the object has type *fstream* or *string* the data will be delivered exactly as found in the PDF file, with the exception of ASCII85 and ASCIIHex filters which will be removed.

Details This function will throw an exception if pCOS does not run in full mode. As an exception, the object */Root/Metadata* can also be retrieved in restricted pCOS mode if *nocopy=false* or *plainmetadata=true*. An exception will also be thrown if *path* does not point to an object of type *stream*, *fstream*, or *string*.

Despite its name this function can also be used to retrieve objects of type *string*. Unlike *PDF_pcos_get_string( ),* which treats the object as a text string, this function will not modify the returned data in any way. Binary string data is rarely used in PDF, and cannot be reliably detected automatically. The user is therefore responsible for selecting the appropriate function for retrieving string objects as binary data or text.

Scope any

Bindings C and C++ language bindings: The returned data buffer can be used until the next call to this function.

Note *This function can be used to retrieve embedded font data from a PDF. Users are reminded that fonts are subject to the respective font vendor's license agreement, and must not be reused without the explicit permission of the respective intellectual property owners. Please contact your font vendor to discuss the relevant license agreement.*

8.3 Other PDI Processing

C++ Java ***int process_pdi(int doc, int page, String optlist)***

Perl PHP ***int PDF_process_pdi(resource p, int doc, int page, string optlist)***

C ***int PDF_process_pdi(PDF *p, int doc, int page, const char* optlist)***

Process certain elements of an imported PDF document.

doc A valid PDF document handle retrieved with *PDF_open_pdi_document( )*.

page If *optlist* requires a page handle (see Table 8.4), *page* must be a valid PDF page handle (not a page number!) retrieved with *PDF_open_pdi_page( )*. The page handle must not have been closed. If *optlist* does not require any page handle, *page* must be -1.

optlist An option list specifying processing options according to Table 8.4. The following options can be used: *action, errorpolicy*

Returns The value 1 if the function succeeded, or an error code of -1 (in PHP: 0) if the function call failed.

Details PDF/X: the output intent must be set either using this function with the *copyoutputintent* option, or with *PDF_load_iccprofile( )*.

PDF/A: the output intent can be set using this function with the *copyoutputintent* option, or with *PDF_load_iccprofile( )*. However, if only device-independent colors are used in the document no output intent is required.

Scope *document*

Table 8.4 Options for PDF_process_pdi( )

key	explanation
action[1]	*(Keyword; required) Specifies the kind of PDF processing:* ***copyoutputintent*** *Copy the PDF/X or PDF/A output intent ICC profile of the imported document to the output document. The second and subsequent attempts to copy an output intent will be ignored. If the document contains more than one output intent the first one will be used. Standard output intents (without an embedded ICC profile) cannot be copied with this method.*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
pdiwarning	*Deprecated, use* `errorpolicy`

1. Does not require a page handle

8.4 Deprecated PDI Parameters

All functions and parameters in this section are deprecated; use the pCOS interface discussed in Section 8.2, »pCOS Functions«, page 127.

C++ Java ***double get_pdi_value(String key, int doc, int page, int reserved)***

Perl PHP ***double PDF_get_pdi_value(resource p, string key, int doc, int page, int reserved)***

C ***double PDF_get_pdi_value(PDF *p, const char *key, int doc, int page, int reserved)***

Deprecated, use *PDF_pcos_get_number( )* with the pCOS paths listed in Table 8.5.

Table 8.5 Names of deprecated numerical parameters for PDF_get_pdi_value( ) and recommended pCOS paths

old key	recommended pCOS path as substitute for the old key
vdp/blockcount	length:pages[...]/PieceInfo/PDFlib/Private/Blocks
/Root/Pages/Count	length:pages
width, height	pages[...]/width, pages[...]/height
/Rotate	pages[...]/Rotate
version	pdfversion (not version!)
/CropBox, /BleedBox, /ArtBox, /TrimBox, /MediaBox	pages[...]/CropBox[0] for llx, pages[...]/CropBox[1] for lly, pages[...]/CropBox[2] for urx, pages[...]/CropBox[3] for ury pages[...]/BleedBox[0] for llx, etc.
vdp/Blocks/<name>/<property>	pages[...]/PieceInfo/PDFlib/Private/Blocks/<name>/<property>
vdp/Blocks[...]/<property>	pages[...]/PieceInfo/PDFlib/Private/Blocks[...]/<property>
vdp/Blocks/<name>/Custom/<property>	pages[...]/PieceInfo/PDFlib/Private/Blocks/<name>/Custom/<property> or pages[...]/PieceInfo/PDFlib/Private/Blocks[...]/Custom/<property>

C++ Java ***String get_pdi_parameter(String key, int doc, int page, int reserved)***

Perl PHP ***string PDF_get_pdi_parameter(resource p, string key, int doc, int page, int reserved)***

C ***const char * PDF_get_pdi_parameter(PDF *p, const char *key, int doc, int page, int reserved, int *len)***

Deprecated, use *PDF_pcos_get_string( )* with the pCOS paths listed in Table 8.6.

Table 8.6 Names of deprecated string parameters PDF_get_pdi_parameter( ) and recommended pCOS paths

old key	recommended pCOS path as substitute for the old key
isempty	pages[...]/isempty
filename	filename
/Info/<key>	/Info/Title etc.
tagged	tagged
pdfx	pdfx
vdp/Blocks/<name>/<property>	pages[...]/PieceInfo/PDFlib/Private/Blocks/<name>/<property>

Table 8.6 Names of deprecated string parameters PDF_get_pdi_parameter( ) and recommended pCOS paths

old key	***recommended pCOS path as substitute for the old key***
vdp/Blocks[...]/<property>	*pages[...]/PieceInfo/PDFlib/Private/Blocks[...]/<property>*
vdp/Blocks/<name>/Custom/<property>	*pages[...]/PieceInfo/PDFlib/Private/Blocks/<name>/Custom/<property> or pages[...]/PieceInfo/PDFlib/Private/Blocks[...]/Custom/<property>*

9 Personalization Functions (PPS)

The PDFlib Personalization Server (PPS) offers dedicated functions for processing variable data blocks of type *Text*, *Image*, and *PDF*. These blocks must be contained in the imported PDF page, but will not be retained in the generated output. The imported page must have been placed on the output page with *PDF_fit_pdi_page( )* before using any of the block filling functions. When calculating the block position on the page, the block functions will take into account the scaling options which have been in effect when placing the imported page with *PDF_fit_pdi_page( )*.

Note *The block processing functions discussed in this chapter require the PDFlib Personalization Server (PPS). The PDFlib Block plugin for Adobe Acrobat is required for creating blocks in PDF templates.*

C++ Java ***int fill_textblock(int page, String blockname, String text, String optlist)***

Perl PHP ***int PDF_fill_textblock(resource p, int page, string blockname, string text, string optlist)***

C ***int PDF_fill_textblock(PDF *p,***
int page, const char *blockname, const char *text, int len, const char *optlist)

Fill a text block with variable data according to its properties.

page A valid PDF page handle for a page containing blocks.

blockname (Name string) The name of the block.

text (Content string) The text to be filled into the block, or an empty string if the default text (as defined by block properties) is to be used. If the *handle* option is supplied and contains a valid Textflow handle this parameter will be ignored.

len (C language binding only) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

optlist An option list specifying filling details according to Table 9.1. The following options can be used:

- *boxsize, charref, encoding, escapesequence, font, ignoreorientation, refpoint, showborder, shrinklimit, textformat, textflowhandle*
- If the *font* option is not supplied, all options for *PDF_load_font( )* (see Table 3.4). These options will only be used if both the *fontname* and *encoding* options are supplied. *autocidfont, autosubsetting, capheight, descender, embedding, fontstyle, keepnative, kerning, linegap, metadata, monospace, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*
- If *textflow=true*, all options of *PDF_add/create_textflow( )* (see Table 4.3):
  General options: *errorpolicy*
  Text semantics: *charclass, charmapping, hyphenchar, tabalignchar,*
  Text formatting: *alignment, avoidemptybegin, fixedleading, hortabsize, hortabmethod, lastalignment, leader, leading, leftindent, minlinecount, parindent, rightindent, ruler, tabalignment*
  Controlling the line breaking algorithm: *adjustmethod, avoidbreak, maxspacing, minspacing, nofitlimit, shrinklimit, spreadlimit*
  Options which work as commands: *comment, mark, nextline, nextparagraph, resetfont, return, space*

Font-related options: *encoding, fontname*
Processing inline options lists: *begoptlistchar, endoptlistchar, textlen*

- If *textflow=true*, the following options of *PDF_fit_textflow( )* (see Table 4.6): *featherlimit, firstlinedist, fitmethod, keep, lastlinedist, linespreadlimit, maxlines, minfontsize, orientate, wrap, rotate, showtabs, verticalalign*
- All appearance options for *PDF_fit_textline( )* (see Table 4.1): *charref, charspacing, dasharray, escapesequence, fakebold, fillcolor, font, fontsize, glyphcheck, horizscaling, italicangle, kerning, matchbox, overline, showborder, strikeout, strokecolor, strokewidth, textformat, textrendering, textrise, underline, underlineposition, underlinewidth, wordspacing*

Both of the options *fontname* and *encoding* can be used to select a font. Alternatively, the *font* option can be used to supply a font handle which has been created with an earlier call to *PDF_load_font( )*. If *font* is specified, the *fontname* and *encoding* options will be ignored.

Returns -1 (in PHP: 0) if the named block doesn't exist on the page, the block cannot be filled (e.g. due to font problems), or the block requires a newer PDFlib version for processing; 1 if the block could be processed successfully. If the *textflowhandle* option is supplied a valid Textflow handle will be returned which can be used in subsequent calls.

If the PDF document is found to be corrupt, this function will either throw an exception or return -1 subject to the *errorpolicy* parameter or option.

Details The supplied text will be formatted into the block, subject to the block's properties. If *text* is empty the function will use the block's default text if available, and silently return otherwise. This may be useful to take advantage of other block properties, such as fill or stroke color.

Linking Textflow blocks: If a Textflow doesn't fit into a block, the *textflowhandle* option can be used to connect multiple blocks to a chain so that they hold multiple parts of the same Textflow:

- In the first call a value of -1 (in PHP: 0) must be supplied. The Textflow handle created internally will be returned by *PDF_fill_textblock( )*, and must be stored by the user.
- In the next call the Textflow handle returned in the previous step can be supplied to the *textflowhandle* option (the text supplied in the *text* parameter will be ignored in this case, and should be empty). The block will be filled with the remainder of the Textflow.
- This process can be repeated with more Textflow blocks.
- The returned Textflow handle can be supplied to *PDF_info_textflow( )* in order to determine the results of block filling, e.g. the end position of the text.

This process can be repeated an arbitrary number of times. The user is responsible for deleting the Textflow handle with *PDF_delete_textflow( )* at the end.

Scope *page, template*

C++ Java ***int fill_imageblock(int page, String blockname, int image, String optlist)***

Perl PHP ***int PDF_fill_imageblock(resource p, int page, string blockname, int image, string optlist)***

C ***int PDF_fill_imageblock(PDF *p, int page, const char *blockname, int image, const char *optlist)***

Fill an image block with variable data according to its properties.

page A valid PDF page handle for a page containing blocks.

blockname (Name string) The name of the block.

image A valid image handle for the image to be filled into the block, or -1 if the default image (as defined by block properties) is to be used.

optlist An option list specifying filling details according to Table 9.1. The following options can be used: *boxsize, errorpolicy, ignoreorientation, refpoint, showborder*

Returns -1 (in PHP: 0) if the named block doesn't exist on the page, the block cannot be filled, or the block requires a newer PDFlib version for processing; 1 if the block could be processed successfully. Use *PDF_get_errmsg( )* to get more information about the nature of the problem.

Details The image referred to by the supplied image handle will be placed in the block, subject to the block's properties. If *image* is -1 (in PHP: 0) the function will use the block's default image if available, and silently return otherwise.

If the PDF document is found to be corrupt, this function will either throw an exception or return -1 subject to the *errorpolicy* parameter or option.

Scope *page, template*

C++ Java ***int fill_pdfblock(int page, String blockname, int contents, String optlist)***

Perl PHP ***int PDF_fill_pdfblock(resource p, int page, string blockname, int contents, string optlist)***

C ***int PDF_fill_pdfblock(PDF *p, int page, const char *blockname, int contents, const char *optlist)***

Fill a PDF block with variable data according to its properties.

page A valid PDF page handle for a page containing blocks.

blockname (Name string) The name of the block.

contents A valid PDF page handle for the PDF page to be filled into the block, or -1 if the default PDF page (as defined by block properties) is to be used.

optlist An option list specifying filling details according to Table 9.1. The following options can be used: *boxsize, encoding, errorpolicy, refpoint, showborder*

Returns -1 (in PHP: 0) if the named block doesn't exist on the page, the block cannot be filled, or the block requires a newer PDFlib version for processing; 1 if the block could be processed successfully. Use *PDF_get_errmsg( )* to get more information about the nature of the problem.

Details The PDF page referred to by the supplied page handle *contents* will be placed in the block, subject to the block's properties. If *contents* is -1 (in PHP: 0) the function will use the block's default PDF page if available, and silently return otherwise.

If the PDF document is found to be corrupt, this function will either throw an exception or return -1 subject to the *errorpolicy* parameter or option.

Scope *page, template*

Table 9.1 Options for the PDF_fill_*block( ) functions

key	explanation
boxsize	*(List of floats) Changes the block's width and height to the specified values (expressed as coordinates in the current user coordinate system). Default: as specified in the block's Rect property.*
charref	*(Boolean; only for PDF_fill_textblock( )) See Table 4.1.*
encoding	*(String) Encoding for the font as required by PDF_load_font( ). This option is required for PDF_fill_textblock( ) unless one of the following is true:* *The string in the* `text` *parameter is empty and the* `defaulttext` *property is used.* *The* `font` *option has been supplied.*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
escapesequence	*(Boolean; only for PDF_fill_textblock( )) See Table 4.1.*
glyphwarning	*Deprecated, use* `errorpolicy`
font	*(Font handle; only for PDF_fill_textblock( )) A font handle returned by PDF_load_font( ). Default: none; either* `font` *or* `fontname` *must be supplied.*
fontwarning	*Deprecated, use* `errorpolicy`
ignoreorientation	*(Boolean; only for PDF_fill_imageblock( )) If* `true`*, the orientation tag in TIFF images will be ignored. Default:* `false`
imagewarning	*Deprecated, use* `errorpolicy`
pdiwarning	*Deprecated, use* `errorpolicy`
refpoint	*(List of floats) Moves the lower left corner of the block to the specified point in user coordinates. Default: as specified in the block's Rect property.*
showborder	*(Boolean) If* `true`*, the border of the block will be stroked (using the current graphics state). This may be useful for development and debugging. Default:* `false`
shrinklimit	*(Float or percentage; only for PDF_fill_textblock( )) See Table 4.1.*
textformat	*(String; only for PDF_fill_textblock( ) unless the* `defaulttext` *property is used) The format used to interpret the supplied text. Default:* `auto`
textflowhandle	*(Textflow handle; only for PDF_fill_textblock( ) with* `textflow=true`*) This option can be used for Textflow block chaining. For the first block in a block chain a value of -1 (in PHP: 0) must be supplied; the value returned by this function can be supplied as Textflow handle in subsequent calls with other blocks in the chain. This option will change the default of* `fitmethod` *to* `clip`*.*

almost any property name	*Block property names and values which will be used to override those in the block definition. The following block properties can not be overridden:*
	`Name`, `Description`, `Locked`, `Subtype`, `Type`
	`defaulttext`, `defaultimage`, `defaultpdf`, `defaultpdfpage`
	As an alternative to supplying the fontname property the `font` *option can be used to supply a font handle (fontname will be ignored in this case).*
	Color properties support the following color space keywords: `none`, `gray`, `rgb`, `cmyk`, `spot`, `spotname`.

10 Interactive Features

10.1 Parameters for Interactive Elements

Table 10.1 lists relevant parameter key names for interactive elements (see Section 2.1, »Parameter Handling«, page 11). These parameters are not available in Unicode-aware language bindings.

Table 10.1 String-related keys for PDF_get/set_parameter( )

key	explanation
hypertextencoding	*Encoding for hypertext strings. An empty string is equivalent to* `unicode`. *Default:* `auto`. *Scope: any*
hypertextformat	*Format for hypertext strings. Possible values are* `bytes`, `utf8`, `utf16`, `utf16le`, `utf16be`, *and* `auto`. *Default:* `auto`. *Scope: any*
usehypertextencoding	*If* `true`, *the encoding specified in the* `hypertextencoding` *parameter will also be used for name strings. If* `false`, *the encoding for name strings without UTF-8 BOM is* `host`. *Default:* `false`. *Scope: any*
usercoordinates	*If* `false`, *coordinates for hypertext rectangles will be expected in the default coordinate system; otherwise the current user coordinate system will be used. Default:* `false`. *Scope: any*

10.2 Actions

C++ Java ***int create_action(String type, String optlist)***

Perl PHP ***int PDF_create_action(resource p, string type, string optlist)***

C ***int PDF_create_action(PDF *p, const char *type, const char *optlist)***

Create an action which can be applied to various objects and events.

type The type of the action, specified by one of the following keywords:

- *GoTo:* go to a destination in the current document.
  Options specific for this type: *destination, destname*
- *GoToR:* go to a destination in another (remote) document.
  Options specific for this type: *destination, destname, filename, newwindow*
- *Launch:* (not for PDF/A) launch an application or document.
  Options specific for this type: *defaultdir, filename, newwindow, operation, parameters*
- *URI:* resolve a uniform resource identifier, i.e. jump to an Internet address.
  Options specific for this type: *ismap, url*
- *Hide:* hide or show an annotation or form field.
  Options specific for this type: *hide, namelist*
- *Named:* execute an Acrobat menu item identified by its name.
  Options specific for this type: *menuname*
- *SubmitForm:* send data to a uniform resource locator, i.e. an Internet address (note that submits which require basic authentication don't work in Acrobat).
  Options specific for this type: *canonicaldate, exclude, exportmethod, submitemptyfields, url*

- *ResetForm:* (not for PDF/A) set some or all form fields to their default values.
- *Trans:* (PDF 1.5) update the display using some visual effect. This can be useful to control the display during a sequence of multiple actions.
  Options specific for this type: *duration, transition*
- *ImportData:* (not for PDF/A) import form field values from a file.
- *JavaScript:* (not for PDF/A) execute a script with JavaScript code.
  Options specific for this type: *script, scriptname*
- *SetOCGState:* (PDF 1.5) hide or show layers.
  Options specific for this type: *layerstate, preserveradio*
- *GoTo3DView:* (PDF 1.6) set the current view of a 3D animation.
  Options specific for this type: *3Dview, target*

optlist An option list specifying properties of the action according to Table 10.2. The following options are supported by all action types (see above for additional type-specific options): *errorpolicy, hypertextencoding*

Returns An action handle which can be used to attach actions to objects within the document. The action handle can be used until the end of the enclosing *document* scope.

Details This function creates a single action. Various objects (e.g. pages, form field events, bookmarks) can be provided with one or more action, but each action must be generated with a separate call to *PDF_create_action( )*. Using an action multiply for different objects is allowed. Actions are prohibited in all PDF/X modes.

Scope *page, document*. The returned handle can be used until the next call to *PDF_end_document( )*.

Table 10.2 Options for action properties with PDF_create_action( )

option	explanation
3Dview	*(Keyword or 3D view handle;* `GoTo3DView`*; required) Selects the view of the target 3D annotation; One of the keywords* `first`, `last`, `next`, `previous`*, (referring to the respective entries in the annotation's* `views` *option), or* `default` *(referring to the annotation's* `defaultview` *option), or a 3D view handle created with PDF_create_3dview( ).*
actionwarning	*Deprecated, use* `errorpolicy`
canonical-date	*(Boolean;* `SubmitForm`*) If* `true`*, any submitted field values representing dates are converted to a standard format. The interpretation of a field as a date is not specified explicitly in the field itself, but only in the JavaScript code that processes it. Default:* `false`
defaultdir	*(String;* `Launch`*) Set the default directory for the launched application. This is only supported by Acrobat on Windows. Default:* `none`
destination	*(Option list;* `GoTo`, `GoToR`*; required unless* `destname` *is supplied) Option list according to Table 10.3 defining the destination to jump to.*
destname	*(Hypertext string;* `GoTo`, `GoToR`*; required unless destination is supplied) Name of a destination which has been defined with PDF_add_nameddest( ) (for GoTo), or the name of a destination in the remote document (for GoToR).*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
duration	*(Float;* `Trans`*) Set the duration of the transition effect in seconds for the current page. Default: 1*

Table 10.2 Options for action properties with PDF_create_action( )

option	explanation
exclude	*(Boolean;* SubmitForm*) If* true*, the* namelist *option specifies which fields to exclude; all fields in the document are submitted except those listed in the* namelist *array and those whose exportable option is false. If false, the* namelist *option specifies which fields to include in the submission. All members of specified field groups will be submitted as well. Default:* false (ResetForm) *If* true*, the* namelist *option specifies which fields to exclude; all fields in the document are reset except those listed in the* namelist *array. If false, the* namelist *option specifies which fields to include in resetting. All members of specified field groups will be reset as well. Default:* false
exportmethod	*(Keyword list;* SubmitForm*) Controls how the field names and values are submitted. Default:* fdf*.* **html, fdf, xfdf, pdf** *In HTML, FDF, XFDF, or PDF format, respectively* **getrequest** *(Only for html and pdf) Submit using HTTP GET; otherwise HTTP POST* **updates** *(Only for* fdf*) Include all incremental updates contained in the underlying PDF document* **exclurl** *(Only for* fdf*) The submitted FDF will exclude the url string.* **annotfields** *(Only for* fdf*) Include all annotations and fields.* **onlyuser** *(Only for* fdf *and* annotfields*) The submit will include only those annotations whose name matches the name of the current user, as determined by the remote server.* **coordinate** *(Only for* html*) The coordinates of the mouse click that caused the submitform action will be transmitted as part of the form data. The coordinate values are relative to the upper-left corner of the field's rectangle.* *Example for combined options:* exportmethod {fdf updates onlyuser}
filename	*(String;* GoToR, Launch*; required) The name of an external (PDF or other) file or application which will be opened when the action is triggered.* (ImportData*; required): The name of the external file containing forms data.*
hide	*(Boolean;* Hide*) Indicates whether to hide (*true*) or show (*false*) annotations. Default: true*
hypertextencoding	*(Keyword) Specifies the encoding for the supplied text. An empty string is equivalent to unicode. Default: the value of the global* hypertextencoding *parameter*
ismap	*(Boolean;* URI*) If* true*, the coordinates of the mouse position will be added to the target URI when the url is resolved. Default:* false
layerstate	*(Option list;* SetOCGState*; required) List of pairs where each pair consists of a keyword and a layer handle. The following keywords are supported:* **on** *Activate the layer* **off** *Deactivate the layer* **toggle** *Reverse the state of the layer. If this is used* preserveradio *should be set to false.*

menuname	*(String;* Named*; required) The name of the menu item to be performed. In PDF/A mode only the well-known names* nextpage*,* prevpage*,* firstpage*,* lastpage *are allowed. Otherwise more names will be accepted. To find the names of other menu items you can execute the following code in Acrobat's JavaScript console or debugger:*
	`function MenuList(m, level) {`
	`  console.println(m.cName);`
	`  if (m.oChildren != null)`
	`  for (var i = 0; i < m.oChildren.length; i++)`
	`  MenuList(m.oChildren[i], level + 1);`
	`}`
	`var m = app.listMenuItems();`
	`for (var i=0; i < m.length; i++)`
	`  MenuList(m[i], 0);`

Table 10.2 Options for action properties with PDF_create_action( )

option	explanation
namelist	*(List of strings;* Hide*; required) The names (including group names) of the annotations or fields to be hidden or shown.* (SubmitForm) *The names (including group names) of form fields to include in the submission or which to exclude, depending on the setting of the* exclude *option. Default: all fields are submitted except those whose* exportable *option is false.* (ResetForm) *The names (including group names) of form fields to include in the resetting or which to exclude, depending on the setting of the* exclude *option. Default: all fields are reset.*
newwindow	*(Boolean;* GoToR, Launch*) A flag specifying whether to open the destination document in a new window. If this flag is* false*, the destination document will replace the current document in the same window. Launch: This entry is ignored if the file is not a PDF document. Default: Acrobat behaves according to the current user preference.*
operation	*(Keyword;* Launch*) A keyword specifying the operation to be applied to the document specified in the* filename *option. This is only supported by Acrobat on Windows. If the* filename *option designates an application instead of a document, this option will be ignored and the application is launched. Default:* open. **open** *open a document* **print** *print a document*
parameters	*(String;* Launch*) A parameter string to be passed to the application specified with the* filename *option. This is only supported by Acrobat on Windows. Multiple parameters can be separated with a space character, but individual parameters must not contain any space characters. This option should be omitted if* filename *designates a document. Default: none*
preserve-radio	*(Boolean;* SetOCGState*) If* true*, preserve the radio-button state relationship between layers. Default:* true
script	*(Hypertext string;* JavaScript*; required) A string containing the JavaScript code to be executed.*
scriptname	*(Hypertext string;* JavaScript*) If present, the JavaScript supplied in the* script *option will be inserted as a document-level JavaScript with the supplied name. If the same* scriptname *is supplied more than once in a document only the last script will be used, the others will be ignored. Document-level JavaScript will be executed after loading the document in Acrobat. This may be useful for scripts which are used in form fields.*
submit-emptyfields	*(Boolean;* SubmitForm*; PDF 1.4) If* true*, all fields characterized by the* namelist *and* exclude *options are submitted, regardless of whether they have a value. For fields without a value, only the field name is transmitted. If false, fields without a value are not submitted. Default:* false
target	*(String;* GoTo3DView*; required) Name of the target 3D annotation as specified in the* name *option of PDF_create_annotation( ).*
transition	*(Keyword;* Trans*) Set the transition effect; see Table 2.9 for a list of keywords. Default:* replace
url	*(String;* URI*; required) A Uniform Resource Locator encoded in 7-bit ASCII or EBCDIC (but only containing ASCII characters) specifying the link target. It can point to an arbitrary (Web or local) resource, and must start with a protocol identifier (such as http://) The* textx/texty*,* currentx/currenty*, and* imagewidth/imageheight *parameters may be useful for retrieving positioning information for calculating the dimension of link rectangles.* *(*SubmitForm*; required) A URL specification giving the uniform resource locator (address) of the script at the Web server that will process the submission.*

10.3 Named Destinations

C++ Java ***void add_nameddest(String name, String optlist)***

Perl PHP ***PDF_add_nameddest(resource p, string name, string optlist)***

C ***void PDF_add_nameddest(PDF *p, const char *name, int len, const char *optlist)***

Create a named destination on an arbitrary page in the current document.

name (Hypertext string) The name of the destination, which can be used as a target for links, bookmarks, or other triggers. Destination names must be unique within a document. If the same name is supplied more than once for a document only the last definition will be used, the others will be silently ignored.

len (C language binding only) Length of *name* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

optlist An option list specifying the destination according to Table 10.3. An empty list is equivalent to *{type fitwindow page 0}*. The following options can be used:
bottom, group, hypertextencoding, hypertextformat, left, page, right, top, type, zoom

Details The destination details must be specified in *optlist*, and the destination may be located on any page in the current document. The provided *name* can be used with the *destname* option in *PDF_create_action( )*, *PDF_create_annotation( )*, *PDF_create_bookmark( )*, and *PDF_begin/end_document( )*. This way defining and using a destination can be split into two separate steps.

Alternatively, if the destination is known at the time when it is used, defining and using the named destination can be combined by using the *destination* option of those functions, and *PDF_add_nameddest( )* is not required in this case.

Scope *document, page*

Table 10.3 Destination options for PDF_add_nameddest( ), as well as for the `destination` *option in PDF_create_action( ), PDF_create_annotation( ), PDF_create_bookmark( ), and PDF_begin/end_document( ).*

option	explanation
bottom	*(Float; only for* `type=fitrect`*) The y coordinate of the page which will positioned at the bottom edge of the window. Default: 0*
group	*(String; required if the* `page` *option has been specified and the document uses page groups; not allowed otherwise.) Name of the page group that the destination page belongs to.*
hypertext-encoding	*(Keyword) Specifies the encoding for the* `name` *parameter. An empty string is equivalent to unicode. Default: the value of the global* `hypertextencoding` *parameter*
hypertext-format	*(Keyword) Sets the format for the* `name` *parameter. Possible values are* `bytes`*,* `utf8`*,* `utf16`*,* `utf16le`*,* `utf16be`*, and* `auto`*. Default: the value of the* `hypertextformat` *parameter*
left	*(Float; only for* `type=fixed`*,* `fitheight`*,* `fitrect`*, or* `fitvisibleheight`*) The x coordinate of the page which will positioned at the left edge of the window. Default: 0*
page	*(Integer) Page number of the destination page (first page is 1). The page must exist in the destination PDF. Page 0 means the current page if in scope page, and page 1 if in scope document. Note that due to a bug Acrobat 6.0 will ignore the page number, and will always jump to page 1. This bug has been fixed in Acrobat 6.0.1, and is not present in older versions. Default: 0*

Table 10.3 Destination options for PDF_add_nameddest( ), as well as for the destination *option in PDF_create_action( ), PDF_create_annotation( ), PDF_create_bookmark( ), and PDF_begin/end_document( ).*

option	***explanation***
right	*(Float; only for* type=fitrect*) The x coordinate of the page which will positioned at the right edge of the window. Default: 1000*
top	*(Float; only for* type=fixed*,* fitwidth*,* fitrect*, or* fitvisiblewidth*) The y coordinate of the page which will positioned at the top edge of the window. Default: 1000*
type	*(Keyword) Specifies the location of the window on the target page. Default:* fitwindow*.* ***fixed*** *Use a fixed destination view specified by the* left*,* top*, and* zoom *options. If any of these is missing its current value will be retained.* ***fitwindow*** *Fit the complete page to the window.* ***fitwidth*** *Fit the page width to the window, with the y coordinate top at the top edge of the window.* ***fitheight*** *Fit the page height to the window, with the x coordinate left at the left edge of the window.* ***fitrect*** *Fit the rectangle specified by left, bottom, right, and top to the window.* ***fitvisible*** *Fit the visible contents of the page (the ArtBox) to the window.* ***fitvisiblewidth*** *Fit the visible contents of the page to the window with the y coordinate top at the top edge of the window* ***fitvisibleheight*** *Fit the visible contents of the page to the window with the x coordinate left at the left edge of the window.*
zoom	*(Float or percentage; only for* type=fixed*) The zoom factor (1 means 100%) to be used to display the page contents. If this option is missing or 0 the zoom factor which was in effect when the link was activated will be retained.*

10.4 Annotations

C++ Java ***void create_annotation(double llx, double lly, double urx, double ury, String type, String optlist)***

Perl PHP ***PDF_create_annotation(resource p, float llx, float lly, float urx, float ury, string type, string optlist)***

C ***void PDF_create_annotation(PDF *p,***
double llx, double lly, double urx, double ury, const char *type, const char *optlist)

Create a rectangular annotation on the current page.

llx, lly, urx, ury *x* and *y* coordinates of the lower left and upper right corners of the annotation rectangle in default coordinates (if the *usercoordinates* parameter or option is *false)* or user coordinates (if it is *true)*. Acrobat will align the upper left corner of the annotation at the upper left corner of the specified rectangle.

Note that annotation coordinates are different from the parameters of the *PDF_rect( )* function. While *PDF_create_annotation( )* expects parameters for two corners directly, *PDF_rect( )* expects the coordinates of one corner, plus width and height values.

If the *usematchbox* option has been specified, the *llx/lly/urx/ury* parameters will be ignored.

type The type of the annotation, specified by one of the following keywords:

- *3D:* (PDF 1.6) animated 3D model
  Options specific for this type: *3Dactivate, 3Ddata, 3Dinteractive, 3Dshared, 3Dinitialview*
- *Circle:* circle annotation
  Options specific for this type: *interiorcolor*
- *FileAttachment:* (not for PDF/A) file attachment annotation. Acrobat Reader 5 is unable to deal with file attachments and will display a question mark instead. File attachments only work in the full Acrobat software.
  Options specific for this type: *filename, iconname, mimetype*
- *FreeText:* free text annotation
  Options specific for this type: *alignment, fillcolor, font, fontsize, orientate*
- *Highlight:* highlight annotation
  Options specific for this type: *polylinelist*
- *Ink:* ink annotation
  Options specific for this type: *polylinelist*
- *Line:* line annotation
  Options specific for this type: *endingstyles, interiorcolor, line*
- *Link:* link annotation
  Options specific for this type: *destination, destname, highlight*
- *Polygon:* (PDF 1.5) polygon annotation (vertices connected by straight lines)
  Options specific for this type: *polylinelist*
- *PolyLine:* (PDF 1.5) polyline annotation; similar to polygons, except that the first and last vertices are not connected.
  Options specific for this type: *endingstyles, interiorcolor, polylinelist*
- *Popup:* pop-up annotation
  Options specific for this type: *open, parentname*
- *Square:* square annotation
  Options specific for this type: *interiorcolor*
- *Squiggly:* (PDF 1.4) squiggly-underline annotation
  Options specific for this type: *polylinelist*

- *Stamp:* rubber stamp annotation
  Options specific for this type: *iconname, orientate*
- *StrikeOut:* strikeout annotation
  Options specific for this type: *polylinelist*
- *Text:* text annotation. In Acrobat this type is called *note* annotation.
  Options specific for this type: *iconname, open*
- *Underline:* underline annotation
  Options specific for this type: *polylinelist*

optlist An option list specifying annotation properties according to Table 10.4. The following options are supported by all annotation types (see above for additional type-specific options):
action, annotcolor, borderstyle, cloudy, contents, createdate, custom, dasharray, display, hypertextencoding, layer, linewidth, locked, name, opacity, popup, readonly, rotate, subject, title, usematchbox, usercoordinates, zoom

Details In all PDF/X modes annotations are only allowed if they are positioned completely outside of the BleedBox (or TrimBox/ArtBox if no BleedBox is present).

Tagged PDF: the annotation will be inserted as a child of the current item if an item is currently active.

Scope *page*

Table 10.4 Options for PDF_create_annotation( )

option	explanation
3Dactivate	*(Option list; only for* `type=3D`*) Specifies when the 3D annotation should be activated and its state upon activation/deactivation. Supported options:* ***enable*** *(Keyword) Specifies when the annotation should be enabled. Default:* `click`. ***open*** *Activate when the page is opened.* ***visible*** *Activate when the page becomes visible.* ***click*** *Annotation must explicitly be activated by a script or user action.* ***enablestate*** *(Keyword) Initial animation state. Default:* `play`. ***pause*** *The 3D model is instantiated, but script animations are disabled.* ***play*** *The 3D model is instantiated; script animations are enabled if present.* ***disable*** *(Keyword) Specifies when the annotation should be disabled. Default:* `invisible`. ***close*** *Deactivate when the page is closed.* ***invisible*** *Deactivate when the page becomes invisible.* ***click*** *Annotation must explicitly be deactivated by a script or user action.* ***disablestate*** *(Keyword) State of the annotation upon disabling. Default:* `reset`. ***pause*** *The 3D model can be rendered, but animations are disabled.* ***play*** *The 3D model can be rendered and animations are enabled.* ***reset*** *Initial state of the 3D model before it has been used in any way.*
3Ddata	*(Option list; only for* `type=3D`*; required) 3D handle created with PDF_load_3ddata( ).*
3Dinteractive	*(Boolean; only for* `type=3D`*) If* `true`*, the 3D model is intended for interactive use. If* `false`*, it is intended to be manipulated with JavaScript. Default:* `true`
3Dshared	*(Boolean; only for* `type=3D`*) If* `true`*, the 3D data specified in the* `3Ddata` *option will be referenced indirectly. Multiple 3D annotations which indirectly reference the same data share a single run-time instance of the model. This means that changes will be visible in all such annotations simultaneously. Default:* `false`
3Dinitialview	*(Keyword or 3D view handle) Specifies the initial view of the 3D model; One of the keywords* `first`, `last`, *(referring to the respective entries in the model's* `views` *option), or* `default` *(referring to the model's* `defaultview` *option), or a 3D view handle created with PDF_create_3dview( ). Default:* `default`

Table 10.4 Options for PDF_create_annotation( )

option	explanation
action	*(Action list) List of annotation actions for the following event. Default: empty list.* ***activate*** *Actions to be performed when the annotation is activated. All types of actions are permitted.*
alignment	*(Keyword; only for* `type=FreeText`*) Alignment of text in the annotation: left, center, right. This option does not work in Acrobat 6, which always uses left. Default:* `left`
annotcolor	*(Color) The color of the background of the annotation's icon when closed, the title bar of the annotation's pop-up window, and the border of a link annotation. Supported color spaces: none, gray, rgb. Default: none* *In PDF/A mode this option must only be used if an RGB output intent has been specified.*
annotwarning	*Deprecated, and not required*
borderstyle	*(Keyword) Style of the annotation border or the line of the annotation types Polygon, PolyLine, Line, Square, Circle, Ink: solid, beveled, dashed, inset, underline. Note that the beveled, inset, and underline styles do not work reliably in Acrobat. Default:* `solid`
cloudy	*(Float; only for* `type=Polygon`*; PDF 1.5) Specifies the intensity of the »cloud« effect used to render the polygon. Possible values are 0 (no effect), 1, and 2. If this option is used the* `borderstyle` *option will be ignored. Default: 0*
contents	*(Hypertext string with a maximum length of 65535 bytes) Text to be displayed for the annotation or (if the annotation does not display text) an alternate description of its contents in human-readable form. Carriage return or line feed characters can be used to force a new paragraph.* *This option is required – but may be an empty string – for* `type=Circle`, `FileAttachment`, `FreeText`, `Highlight`, `Ink`, `Line`, `Polygon`, `PolyLine`, `Square`, `Squiggly`, `Stamp`, `Strikeout`, `Text`, `Underline`. *If* `type=FreeText` *this option must be of type string. This option is optional for* `type=Link`, `PopUp`. *In PDF/A-1a mode this option is required (and must contain a non-empty string).*
createdate	*(Boolean; PDF 1.5 or above) If* `true`*, a date/time entry will be created for the annotation. Default:* `false`
custom	*(List of option lists; only for advanced users) This option can be used to insert an arbitrary number of private entries in the annotation dictionary, which may be useful for specialized applications such as inserting processing instructions for digital printing machines. Using this option requires knowledge of the PDF file format and the target application. Corrupt PDF output may be generated if unsuitable values are supplied. Each list must contain three options:* ***key*** *(string) Name of the dictionary key (excluding the / character). Any non-standard PDF key can be specified, as well as the following standard keys:* `Contents`, `Name` *(option* `iconname`*),* `NM` *(option* `name`*), and* `Open`*. The corresponding options will be ignored in this case.* ***type*** *(keyword) Type of the corresponding value, which must be one of* `boolean`, `name`, *or* `string` ***value*** *(Hypertext string if* `type=string`*, otherwise string) Value as it will appear in the PDF output; PDFlib will automatically apply any decoration required for strings and names.*
dasharray	*(List of floats; only for* `borderstyle=dashed`*). The lengths of dashes and gaps for a dashed border in default units (see PDF_setdash( )). Default: 3 3*
destination	*(Option list; only for* `type=Link`*; will be ignored if an activate action has been specified) Option list according to Table 10.3 defining the destination to jump to.*
destname	*(Hypertext string; only for* `type=Link`*; will be ignored if the* `destination` *option has been specified) Name of a destination which has been defined with PDF_add_nameddest( ). Destination or* `destname` *actions are dominant over this option.*
display	*(Keyword) Visibility on screen and paper:* `visible`, `hidden`, `noview`, `noprint`. *Default:* `visible`
endingstyles	*(Keyword list; only for* `type=Line`, `PolyLine`*) A list with two keywords specifying the line ending styles:* `none`, `square`, `circle`, `diamond`, `openarrow`, `closedarrow`. *Default:* {`none none`}

Table 10.4 Options for PDF_create_annotation( )

option	explanation
filename	*(String; only for* type=FileAttachment*; required) The file associated with the annotation. It is recommended to use only ASCII characters in the filename.*
fillcolor	*(Color; only for* type=FreeText*) Fill color of the text. Supported color spaces: gray, rgb, cmyk. Default:* {gray 0} *(=black)* *In PDF/A mode this option must only be used if an RGB or CMYK output intent has been specified, and a corresponding rgb or cmyk color space must be used.*
font	*(Font handle; only for* type=FreeText*; required) Specifies the font to be used for the annotation. Only PDF core fonts and the following encodings are allowed: any 8-bit encoding, Unicode CMaps,* builtin.
fontsize	*(Float or option list; only for* type=FreeText*; required) The font size in default or user coordinates depending on the* usercoordinates *option or parameter. See PDF_fit_textline( ) for details.*
highlight	*(Keyword; only for* type=Link*) Highlight mode of the annotation when the user clicks on it:* none, invert, outline, push. *Default:* invert
hypertext-encoding	*(Keyword) Specifies the encoding for the supplied text. An empty string is equivalent to unicode. Default: the value of the global* hypertextencoding *parameter*
iconname	*(String; only for* type=Text, Stamp, FileAttachment*) Name of an icon to be used in displaying the annotation (to create an annotation without any visible icon set opacity=0):* *For* type=Text *(default: note):* *comment*  *, key*  *, note*  *, help*  *, newparagraph*  *, paragraph*  *, insert*  *For* type=Stamp *(default: draft):* *approved, experimental, notapproved, asis, expired, notforpublicrelease, confidential, final, sold, departmental, forcomment, topsecret, draft, forpublicrelease.* *For* type=FileAttachment *(default: pushpin):* *graph*  *, pushpin*  *, paperclip*  *, tag* 
interiorcolor	*(Color; only for* type=Line, PolyLine, Square, Circle*) The color for the annotation's line endings, rectangle, or ellipse, respectively. Supported color spaces: none, gray, rgb. Default:* none *In PDF/A mode this option must only be used if an RGB output intent has been specified.*
layer	*(Layer handle; PDF 1.5) Layer to which the annotation will belong. The annotation will only be visible if the corresponding layer is visible.*
line	*(List of 4 floats; only for* type=Line*; required) A list of four numbers x1, y1, x2, y2 specifying the start and end coordinates of the line in default coordinates (if the* usercoordinates *parameter is* false*) or user coordinates (if it is* true*).*
linewidth	*(Integer) Width of the annotation border or the line of the annotation types* Line, PolyLine, Polygon, Square, Circle, Ink *in default units (=points). If linewidth = 0 the border will be invisible. Default: 1*
locked	*(Boolean) If* true*, the annotation properties cannot be edited in Acrobat. Default:* false
mimetype	*(String; only for* type=FileAttachment*) MIME type of the file. Acrobat will use it for launching the appropriate application when the annotation is activated.*
name	*(String) Name uniquely identifying the annotation. The name is necessary for some actions, and must be unique on the page. Default: none*
opacity	*(Float or percentage; PDF 1.4) The constant opacity value (0-1 or 0%-100%) to be used in painting the annotation. Default: 1*
open	*(Boolean; only for* type=Text, Popup*) If* true*, the annotation will initially be displayed open. Default:* false

Table 10.4 Options for PDF_create_annotation( )

option	explanation
orientate	*(Keyword; only for* `type=FreeText, Stamp`*) Specifies the desired orientation of the annotation within its rectangle. Default:* `north` ***north*** *upright* ***east*** *pointing to the right* ***south*** *upside down* ***west*** *pointing to the left*
parentname	*(String; only for* `type=PopUp`*) Name of the parent annotation for the annotation.*
polylinelist	*(List containing one or more lists of floats; only for* `type=Polygon, PolyLine, Ink, Highlight, Underline, Squiggly, Strikeout`*). The coordinates will be interpreted in default coordinates (if the* `usercoordinates` *option is* `false`*) or user coordinates (if it is* `true`*). Default: a polyline connecting the vertices of the annotation rectangle.* ***type=Polygon, PolyLine, Ink*** *A single list containing a polyline with n segments (minimum: n=2). A polyline is a list of 2 x n float values specifying coordinate pairs. The points will be connected by straight lines. Example for n=3:* `{{10 20 30 40 50 60 }}` ***others*** *The list contains n sublists with 8 float values each, specifying n quadrilaterals (minimum: n=1). Each quadrilateral encompasses a word or group of contiguous words in the text underlying the annotation. The coordinates for each quadrilateral are given as x4 y4 x3 y3 x1 y1 x2 y2 specifying the quadrilateral's vertices in counterclockwise order (x4 y4 is the upper left corner). The text is oriented with respect to the edge connecting (x1, y1) and (x2, y2). Example for n=2:* `{{1 2 3 4 5 6 7 8} {10 20 30 40 50 60 70 80}}`
popup	*(String) Name of a PopUp annotation for entering or editing the text associated with this annotation. Default: none*
readonly	*(Boolean) If* `true`*, do not allow the annotation to interact with the user. The annotation may be displayed or printed, but should not respond to mouse clicks or change its appearance in response to mouse motions. Default:* `false`
rotate	*(Boolean; must not be set to* `true` *in PDF/A mode) If* `true`*, rotate the annotation to match the rotation of the page. Otherwise the annotation's rotation will remain fixed. This option will be ignored for the icons of text annotations. Default:* `false` *in PDF/A mode,* `true` *otherwise*
subject	*(Hypertext string; PDF 1.5) Text representing a short description of the subject being addressed by the annotation. Default: none*
title	*(Hypertext string) The text label to be displayed in the title bar of the annotation's pop-up window when open and active (in Acrobat it will be labeled as »Author«). The maximum length of title is 255 single-byte characters or 126 Unicode characters. However, a practical limit of 32 characters for title is advised. This string corresponds to the »Author« field in Acrobat. Default: none*
usematchbox	*(List of strings) The* `llx/lly/urx/ury` *parameters will be ignored, and the matchbox will be used instead. The first element in the option list is a name string which specifies a matchbox. The second element is either an integer specifying the number of the desired rectangle, or the keyword* `all` *to specify all rectangles referring to the selected matchbox. If the second element is missing, it defaults to* `all`*.* *If the matchbox itself or the specified rectangle does not exist on the current page, the function will silently return without creating any annotation.*
user-coordinates	*(Boolean) If false, annotation coordinates and font size will be expected in the default coordinate system; otherwise the current user coordinate system will be used. Default: the value of the global* `usercoordinates` *parameter*
zoom	*(Boolean; must not be set to* `true` *in PDF/A mode) If* `true`*, scale the annotation to match the magnification of the page. Otherwise the annotation's size will remain fixed. This option will be ignored for the icons of text annotations. Default:* `false` *in PDF/A mode,* `true` *otherwise*

10.5 Form Fields

C++ Java ***void create_field(double llx, double lly, double urx, double ury, String name, String type, String optlist)***

Perl PHP ***PDF_create_field(resource p, float llx, float lly, float urx, float ury, string name, string type, string optlist)***

C ***void PDF_create_field(PDF *p, double llx, double lly, double urx, double ury, const char *name, int len, const char *type, const char *optlist)***

Create a form field on the current page subject to various options.

llx, lly, urx, ury *x* and *y* coordinates of the lower left and upper right corners of the field rectangle in default coordinates (if the *usercoordinates* parameter or option is *false)* or user coordinates (if it is *true)*.

Note that form field coordinates are different from the parameters of the *PDF_rect( )* function. While *PDF_create_field( )* expects parameters for two corners directly, *PDF_rect( )* expects the coordinates of one corner, plus width and height values.

name (Hypertext string) The form field name, possibly prefixed with the name(s) of one or more groups which have been created with *PDF_create_fieldgroup( )*. Group names must be separated from each other and from the field name by a period ».« character. Field names must be unique on a page, and must not end in a period ».« character.

len (C language binding only) Length of *text* (in bytes) for UTF-16 strings. If *len = 0* a null-terminated string must be provided.

type The field type according to Table 10.5.

Table 10.5 Form field types

type	*icon*	*Options specific for this type*
pushbutton		*buttonlayout, caption, captiondown, captionrollover, charspacing, fitmethod, icon, icondown, iconrollover, position, submitname*
checkbox		*currentvalue, itemname*
radiobutton		*buttonstyle, currentvalue, itemname, toggle, unisonselect* *The name must be prefixed with a group name since radio buttons must always belong to a group. For all other field types group membership is optional.*
listbox		*charspacing, currentvalue, itemnamelist, itemtextlist, multiselect, sorted, topindex*
combobox		*commitonselect, charspacing, currentvalue, editable, itemnamelist, itemtextlist, sorted, spellcheck*
textfield		*comb, charspacing, currentvalue, fileselect, maxchar, multiline, password, richtext, scrollable, spellcheck*
signature		*charspacing, lockmode*

optlist An option list specifying the field's properties according to Table 10.6. String options will be interpreted as hypertext strings or text strings as noted in the table. The

following options are supported for all field types (see Table 10.5 for more type-specific options):
action, alignment, backgroundcolor, bordercolor, borderstyle, calcorder, dasharray, defaultvalue, display, errorpolicy, exportable, fieldtype, fillcolor, font, fontsize, highlight, hypertextencoding, hypertextformat, layer, linewidth, locked, orientate, readonly, required, strokecolor, taborder, tooltip, usercoordinates

Details The tab order of the fields on the page (the order in which they receive the focus when the tab key is pressed) is determined by the order of calls to *PDF_create_field( )* by default, but a different order can be specified with the *taborder* option. The tab order can not be modified after creating the fields. However, this behavior can be overridden with the *taborder* option of *PDF_begin/end_page_ext( )*.

In Acrobat it is possible to assign a format (number, percentage, etc.) to text fields. However, this is not specified in the PDF reference, but implemented with custom JavaScript. You can achieve the same effect by attaching JavaScript actions to the field which refers to the predefined (but not standardized) JavaScript functions in Acrobat.

This function must not be called in PDF/A mode.

In all PDF/X modes form fields are only allowed if they are positioned completely outside of the BleedBox (or TrimBox/ArtBox if no BleedBox is present).

Tagged PDF: the field will be inserted as a child of the current item if an item is currently active.

Scope *page*

Table 10.6 Options for field properties with PDF_create_field( ) and PDF_create_fieldgroup( )

option	***explanation***
action	*(Action list) List of field actions for one or more of the following events. The activate event is allowed for all field types, the other events are not allowed for* `type=pushbutton`*,* `checkbox`*, and* `radiobutton`*. Default: empty list.*
	activate *Actions to be performed when the field is activated.*
	keystroke *JavaScript actions to be performed when the user types into a text field or combo box, or modifies the selection in a scrollable list box.*
	format *JavaScript actions to be performed before the field is formatted to display its current value. This allows the field's value to be modified before formatting.*
	validate *JavaScript actions to be performed when the field's value is changed. This allows the new value to be checked for validity.*
	calculate *JavaScript actions to be performed in order to recalculate the value of this field when the value of another field changes.*
	enter *Actions to be performed when the mouse enters the field's area.*
	exit *Actions to be performed when the mouse exits the field's area.*
	down *Actions to be performed when the mouse button is pressed inside the field's area.*
	up *Actions to be performed when the mouse button is released inside the field's area (this is typically used to activate a field).*
	focus *Actions to be performed when the field receives the input focus.*
	blur *Actions to be performed when the field loses the input focus.*
alignment	*(Keyword) Alignment of text in the field:* `left`*,* `center`*,* `right`*. Default:* `left`
backgroundcolor ***bordercolor***	*(Color) Color of the field background or border. Supported color spaces: none, gray, rgb, cmyk. Default:* `none`

Table 10.6 Options for field properties with PDF_create_field( ) and PDF_create_fieldgroup( )

option	explanation
borderstyle	*(Keyword) Style of the field border, which is one of* solid, beveled, dashed, inset, underline. *Default:* solid
button-layout	*(Keyword; only for* type=pushbutton*) The position of the button caption relative to the button icon, provided both have been specified:* below, above, right, left, overlaid. *Default:* right
buttonstyle	*(Keyword; only for* type=radiobutton *and* checkbox*) Specifies the symbol to be used for the field:* check, cross, diamond, circle, star, square. *Default:* check
calcorder	*(Integer; only used if the field has a JavaScript action for the calculate event) Specifies the calculation order of the field relative to other fields. Fields with smaller numbers will be calculated before fields with higher numbers. Default: 10 plus the maximum calcorder used on the current page (and 10 initially)*
caption	*(Content string; only for* type=pushbutton*; one of the* caption *or* icon *options must be supplied for push buttons) The caption text which will be visible when the button doesn't have input focus. Use an empty string (i.e. caption { }) if you don't want caption nor icon. Default: none*
captiondown	*(Content string; only for* type=pushbutton*) The caption text which will be visible when the button is activated. Default: none*
caption-rollover	*(Content string; only for* type=pushbutton*) The caption text which will be visible when the button has input focus. Default: none*
charspacing	*(Float; not for* type=radiobutton *and* checkbox*) The character spacing for text in the field in units of the current user coordinate system. This option is ignored by Acrobat 7. Default: 0*
comb	*(Boolean; only for* type=textfield*; PDF 1.5) If* true *and the* multiline, fileselect, *and* password *options are false, and the* maxchar *option has been supplied with an integer value, the field will be divided into a number of equidistant subfields (according to the maxchar value) for individual characters. Default:* false
commit-onselect	*(Boolean; only for* type=listbox *and* combobox*; PDF 1.5) If* true*, an item selected in the list will be committed immediately upon selection. If false, the item will only be committed upon exiting the field. Default:* false
currentvalue	*(Not for* type=pushbutton *and* signature*) The field's initial value. Type and default depend on the field type:* ***checkbox, radiobutton*** *(String) Arbitrary string other than* Off *means that the button is activated; Acrobat 6 shows erratic behavior if itemname is specified and/or unisonselect is true. The string* Off *means that the button is deactivated. This option should be set for the first button. Default:* Off ***textfield, combobox*** *(Content string) Contents of the field. Default: empty* ***listbox*** *(List of integers) Indices of the selected items within itemtextlist. Default: none*
dasharray	*(List of floats; only for* borderstyle=dashed*). The lengths of dashes and gaps for a dashed border in default units (see PDF_setdash( )). Default: 3 3*
defaultvalue	*The field's value after a reset action. Types and defaults are the same as for the* currentvalue *option. Exception: for listboxes only a single integer value is allowed.*
display	*(Keyword) Visibility on screen and paper:* visible, hidden, noview, noprint. *Default:* visible
editable	*(Boolean; only for* type combobox*) If* true*, the currently selected text in the box can be edited. Default:* false
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
exportable	*(Boolean) The field will be exported when a SubmitForm action happens. Default:* true

Table 10.6 Options for field properties with PDF_create_field( ) and PDF_create_fieldgroup( )

option	explanation
fieldtype	*(Keyword; only for PDF_create_fieldgroup( )) Type of the fields contained in the group:* mixed, pushbutton, checkbox, radiobutton, listbox, combobox, textfield, *or* signature. *Unless* fieldtype=mixed *the group may only contain fields of the specified type. If a particular fieldtype has been specified for the group, the current value is displayed in all contained fields simultaneously, even if the fields are located on separate pages. If* fieldtype=radiobutton *the option* unisonselect *must be supplied. The options* itemtextlist, itemnamelist, currentvalue *and* defaultvalue *must be specified in the field group options, and not in the individual fields' options. Default:* mixed
fieldwarning	*Deprecated, use* errorpolicy
fileselect	*(Boolean; only for* type=textfield*) If* true*, the text in the field will be treated as a file name. Default:* false
fillcolor	*(Color) Fill color for text. Supported color spaces: gray, rgb, cmyk. Default:* {gray 0} *(=black)*
fitmethod	*(Keyword; only for* type=pushbutton*) Method of placing a template provided with the* icon, icondown, *and* iconrollover *options within the button. Default:* meet. **auto** *same as* meet *if the template fits into the button, otherwise* clip **nofit** *same as* clip **clip** *template will not be scaled, but clipped at the field border* **meet** *template will be scaled proportionally so that it fits into the button* **slice** *same as* meet **entire** *template will be scaled so that it fully fits into the button*
font	*(Font handle; required except for* type=radiobutton *and* checkbox *which always use ZapfDingbats). Specifies the font to be used for the field. The following options must have been set in the corresponding call to PDF_load_font( ):* embedding *(with the exception of core fonts which need not be embedded),* nosubsetting, noautocidfont. *Only the following encodings are allowed: 8-bit encodings, any CMaps (but only for standard CJK fonts), builtin*
fontsize	*(Float, option list, or keyword) Font size in user coordinates. If the keyword* auto *is supplied instead of a float value Acrobat will determine the font size automatically. See PDF_fit_textline( ) for details. Default:* auto
highlight	*(Keyword) Highlight mode of the field when the user clicks on it:* none, invert, outline, push. *Default:* invert
hypertext-encoding	*(Keyword) Specifies the encoding for the* name *parameter. An empty string is equivalent to unicode. Default: the value of the global* hypertextencoding *parameter*
hypertext-format	*(Keyword) Sets the format for the* name *parameter. Possible values are bytes, utf8, utf16, utf16le, utf16be, and auto. Default: the value of the* hypertextformat *parameter*
icon	*(Template handle[1]; only for* type=pushbutton*; one of the* caption *or* icon *options must be supplied for push buttons) Handle for a template which will be visible when the button doesn't have input focus. Default: none*
icondown	*(Template handle[1]; only for* type=pushbutton*) Handle for a template which will be visible when the button is activated. Default: none*
iconrollover	*(Template handle[1]; only for* type=pushbutton*) Handle for a template which will be visible when the button has input focus. Default: none*
itemname	*(Hypertext string; only for* type=radiobutton *and* checkbox*; must be used if the export value is not a Latin 1 string) Export value of the field. Item names for multiple radio buttons in a group may be identical. Acrobat 6: Checkboxes within a group which have the same item name will be switched on or off simultaneously, even if they are located on separate pages. Default: field name*

Table 10.6 Options for field properties with PDF_create_field( ) and PDF_create_fieldgroup( )

option	explanation
itemnamelist	*(Hypertext string; only for* type=listbox *and* combobox*) Export values of the list items. Multiple items may have the same export value. Default: none*
itemtextlist	*(List of content strings; only for* type=listbox *and* combobox*, and required in these cases) Text contents for all items in the list. If both* itemnamelist *and* itemtextlist *are specified both must contain the same number of strings.*
layer	*(Layer handle; PDF 1.5) Layer to which the field will belong. The field will only be visible if the corresponding layer is visible.*
linewidth	*(Integer) Line width of the field border in default units (=points). Default: 1*
locked	*(Boolean) If* true*, the field properties cannot be edited in Acrobat. Default:* false
lockmode	*(Keyword; only for* type=signature*; PDF 1.5) Indicates the set of fields that should be locked when the field is signed:* ***all*** *All fields in the document will be locked.*
maxchar	*(Integer or keyword; only for* type=textfield*) The upper limit for the number of text characters in the field, or the keyword* unlimited *if there is no limit. Default:* unlimited
multiline	*(Boolean; only for* type=textfield*) If* true*, text will be wrapped to multiple lines if required. Default:* false
multiselect	*(Boolean; only for* type=listbox*) If* true*, multiple items in the list can be selected. Default:* false
orientate	*(Keyword) Orientation of the contents within the field:* north*,* west*,* south*,* east*. Default:* north
password	*(Boolean; only for* type=textfield*) If* true*, the text will be simulated with bullets or asterisks upon input. Default:* false
position	*(List of floats or keywords; only for* type=pushbutton*) Relative position of a template provided with the icon... options within the button. Default: 50 50*
readonly[2]	*(Boolean) If* true*, the field does not allow any input. Default:* false
required	*(Boolean) If* true*, the field must contain a value when the form is submitted. Default:* false
richtext	*(Boolean; only for* type=textfield*; PDF 1.5) Allow rich text formatting. If true, the fontsize must not be 0, and fillcolor must not use color space cmyk. Default: false*
scrollable	*(Boolean; only for* type=textfield*) If* true*, text will be moved to the invisible area outside the field if the text doesn't fit into the field. If false, no more input will be accepted when the text fills the full field. Default:* true
sorted	*(Boolean; only for* type=listbox *and* combobox*) If* true*, the contents of the list will be sorted. Default:* false
spellcheck	*(Boolean; only for* type=textfield *and* combobox*) If* true*, the spell checker will be active in the field. Default:* true
strokecolor	*(Color) Stroke color for text. Supported color spaces: gray, rgb, cmyk. Default:* {gray 0} *(=black).*
submitname	*(Hypertext string; recommended only for* type=pushbutton*) URL-encoded string of the Internet address to which the form will be submitted. Default: None*
taborder	*(Integer) Specifies the tab order of the field relative to other fields. Fields with smaller numbers will be reached before fields with higher numbers. Default: 10 plus the maximum* taborder *used on the current page (and 10 for the first field on the page); the result of this default is that the creation order will specify the tab order.*

Table 10.6 Options for field properties with PDF_create_field( ) and PDF_create_fieldgroup( )

option	explanation
toggle	*(Boolean; only for PDF_create_fieldgroup( ) and* `type=radiobutton`*) If true, a radio button within a group can be activated and deactivated by clicking. If* `false`*, it can only be activated by clicking, and deactivating by clicking another button. Default:* `false`
tooltip[2]	*(Hypertext string) The text visible in the field's tooltip. For radio buttons and groups Acrobat will always use the tooltip of the first button in the group, others will be ignored. Default: none*
topindex	*(Integer; only for* `type=listbox`*) Index of the first visible entry. The first item has index 0. Default: 0*
unisonselect	*(Boolean; only for PDF_create_fieldgroup( ),* `type=radiobutton` *and PDF 1.5) If* `true`*, radio buttons with the same field name or item name will be selected simultaneously. Default:* `false`
user-coordinates	*(Boolean) If* `false`*, field coordinates will be expected in the default coordinate system; otherwise the current user coordinate system will be used. Default: the value of the global* `usercoordinates` *parameter*

1. Templates for icons can be created with the PDF_begin_template( ) function; if the icon consists of an image only you can create the template by supplying the template option to PDF_load_image( ).
2. For `type=radiobutton` *this option should not be used with PDF_create_field( ), but only with PDF_create_fieldgroup( ).*

C++ Java ***void create_fieldgroup(String name, String optlist)***
Perl PHP ***PDF_create_fieldgroup(resource p, string name, string optlist)***
C ***void PDF_create_fieldgroup(PDF *p, const char *name, int len, const char *optlist)***

Create a form field group subject to various options.

name (Hypertext string) The name of the form field group, which may in turn be prefixed with the name of another group. Field groups can be nested to an arbitrary level. Group names must be separated with a period ».« character. Group names must be unique within the document, and must not end in a period ».« character.

len (C language binding only) Length of *text* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

optlist An option list specifying field properties according to Table 10.6.

Details Field groups are useful for mirroring the contents of a field in one or more other fields. If the name of a field group is provided as prefix for a field name created with *PDF_create_field( )*, the new field will be part of this group. All field property options provided in the *optlist* for a group will be inherited by all fields belonging to this group.

Scope *page, document*

10.6 Bookmarks

C++ Java ***int create_bookmark(String text, String optlist)***

Perl PHP ***int PDF_create_bookmark(resource p, string text, string optlist)***

C ***int PDF_create_bookmark(PDF *p, const char *text, int len, const char *optlist)***

Create a bookmark subject to various options.

text (Hypertext string) Contains the text of the bookmark. The maximum length of *text* is 255 single-byte characters (8-bit encodings), or 126 Unicode characters. However, a practical limit of 32 characters for *text* is recommended.

len (C language binding only) Length of *text* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

optlist An option list specifying the bookmark's properties according to Table 10.7. The following options can be used:
action, destination, destname, fontstyle, hypertextencoding, hypertextformat, index, open, parent, textcolor

Returns A handle for the generated bookmark, which may be used with the *parent* option in subsequent calls.

Details This function adds a PDF bookmark with the supplied *text*. Unless the *destination* option has been specified the bookmark will point to the current page (or the last page if used in *document* scope, or the first page if used before the first page).

Creating bookmarks sets the *openmode* option of *PDF_begin/end_document( )* to *bookmarks* unless another mode has explicitly been set.

Scope *document, page*

Table 10.7 Options for PDF_create_bookmark( )

option	explanation
action	*(Action list) List of bookmark actions for the following event. Default:* `GoTo` *action with the target specified in the* `destination` *option.* ***activate*** *Actions to be performed when the bookmark is activated. All types of actions are permitted.*
destination	*(Option list; will be ignored if an activate action has been specified) Option list specifying the bookmark destination according to Table 10.3. Default: {type fitwindow page 0} if* `destination`*,* `destname`*, and* `action` *are absent.*
destname	*(Hypertext string; May be empty; will be ignored if the* `destination` *option has been specified) Name of a destination which has been defined with PDF_add_nameddest( ). Destination or destname actions will be dominant over this option. If* `destname` *is an empty string (i.e. {}) and neither* `destination` *nor* `action` *are specified, the bookmark won't have any action, which may be useful if the bookmark serves as a separator.*
fontstyle	*(Keyword) Specifies the font style of the bookmark text:* `normal`*,* `bold`*,* `italic`*,* `bolditalic`*. Default:* `normal`
hypertext-encoding	*(Keyword) Specifies the encoding for the supplied text. An empty string is equivalent to unicode. Default: the value of the global* `hypertextencoding` *parameter*
hypertext-format	*(Keyword) Set the format for the supplied text. Possible values are bytes, utf8, utf16, utf16le, utf16be, and auto. Default: the value of the global* `hypertextformat` *parameter*

Table 10.7 Options for PDF_create_bookmark( )

option	explanation
index	*(Integer) Index where to insert the bookmark within the parent. Values between 0 and the number of bookmarks of the same level will be used to insert the bookmark at that specific location within the parent. The value -1 can be used to insert the bookmark as the last one on the current level. Default: -1. However, for inserted or resumed pages bookmarks will be placed as if all pages had been generated in their physical order (the bookmarks will reflect the page order).*
open	*(Boolean) If false, subordinate bookmarks will not be visible. If* `true`*, all children will be folded out. Default:* `false`
parent	*(Bookmark handle) The new bookmark will be specified as a subordinate of the bookmark specified in the handle. If parent = 0 a new top-level bookmark will be created. Default: 0*
textcolor	*(Color) Specifies the color of the bookmark text. Supported color spaces: none, gray, rgb. Default:* `rgb {0 0 0 }` *(=black)*

11 Multimedia Features (3D Artwork)

C++ Java ***int load_3ddata(String filename, String optlist)***

Perl PHP ***int PDF_load_3ddata(string filename, string optlist)***

C ***int PDF_load_3ddata(PDF *p, const char *filename, int len, const char *optlist)***

Load a 3D model from a disk-based or virtual file (requires PDF 1.6).

filename (Name string) Name of a disk-based or virtual file containing a 3D model in U3D format.

len (C language binding only) Length of *filename* (in bytes) for UTF-16 strings. If *len* = 0 a null-terminated string must be provided.

optlist An option list specifying properties of the 3D model according to Table 11.1. The following options can be used: *defaultview, errorpolicy, hypertextencoding, script, views*

Returns A 3D handle which can be used to create 3D annotations with *PDF_create_annotation( )*. The 3D handle can be used until the end of the enclosing *document* scope. The return value must be checked for -1 (in PHP: 0) which signals an error.

Details The file containing 3D data will be loaded. There is no error checking on the 3D data.

Scope *page, document*. The returned handle can be used until the next call to *PDF_end_document( )*.

Table 11.1 Options for PDF_load_3ddata( )

option	explanation
defaultview	*(Keyword or 3D view handle) Specifies the initial view of the 3D annotation; One of the keywords* `first` *or* `last` *(referring to the respective entries in the* `views` *option), or a 3D view handle created with PDF_create_3dview( ). Default:* `first`
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
hypertext-encoding	*(Keyword) Specifies the encoding for the supplied* `script`*. An empty string is equivalent to unicode. Default: the value of the global* `hypertextencoding` *parameter*
script	*(Hypertext string) String containing JavaScript code to be executed when the 3D model is instantiated. Default: no script*
views	*(list of 3D view handles) List of predefined views for the 3D model. Each list element is a 3D view handle created with PDF_create_3dview( ). Default: empty list*

C++ Java ***int create_3dview(String username, String optlist)***

Perl PHP ***int PDF_create_3dview(string username, string optlist)***

C ***int PDF_create_3dview(PDF *p, const char *username, int len, const char *optlist)***

Create a 3D view (requires PDF 1.6).

username (Hypertext string) User interface name of the 3D view.

len (C language binding only) Length of *username* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

optlist An option list specifying 3D view properties according to Table 11.2. The following options can be used: *background, camera2world, cameradistance, errorpolicy, hypertextencoding, name*

Returns A 3D view handle which can be attached to 3D models with the *views* option in *PDF_load_3ddata( )*, to create 3D annotations with *PDF_create_annotation( )*, or to create 3D-related actions with *PDF_create_action( )*. The 3D view handle can be used until the end of the enclosing *document* scope. The return value must be checked for -1 (in PHP: 0) which signals an error.

Details A named 3D view will be created which can be used for loading 3D data in actions

Scope *page, document*. The returned handle can be used until the next call to *PDF_end_document( )*.

Table 11.2 Options for PDF_create_3dview( )

option	explanation
background	*(Option list) Specifies the background for the 3D model:* **fillcolor** *(Color) Background color, expressed in the RGB color space. Default: white* **entire** *(Boolean) If* `true`*, the background applies to the entire annotation; otherwise it applies only to the rectangle specified in the annotations's* `3Dbox` *option. Default:* `false`
camera2world	*(List of 12 floats) 3D transformation matrix specifying position and orientation of the camera in world coordiantes. Default: defined internally in the 3D data*
camera-distance	*(Non-negative float; will be ignored if* `camera2world` *is not specified) Distance between camera and center of orbit. Default: defined internally in the 3D data*
errorpolicy	*(Keyword) Controls the behavior in case of an error (see Section 2.6, »Exception Handling«, page 30)*
hypertext-encoding	*(Keyword) Specifies the encoding for the supplied* `name` *and* `username`*. An empty string is equivalent to unicode. Default: the value of the global* `hypertextencoding` *parameter*
name	*(Hypertext string) Name of the 3D view, used for actions. This is an optional internal name which is treated separately from the required* `username` *parameter.*

12 Document Interchange

12.1 Document Information Fields

C++ Java ***void set_info(String key, String value)***

Perl PHP ***PDF_set_info(resource p, string key, string value)***

C ***void PDF_set_info(PDF *p, const char *key, const char *value)***

C ***void PDF_set_info2(PDF *p, const char *key, const char *value, int len)***

Fill document information field *key* with *value*.

key (Name string) The name of the document info field, which may be any of the standard names, or an arbitrary custom name (see Table 12.1). There is no limit for the number of custom fields. Regarding the use and semantics of custom document information fields, PDFlib users are encouraged to take a look at the Dublin Core Metadata element set.[1]

value (Hypertext string) The string to which the *key* parameter will be set. Acrobat imposes a maximum length of *value* of 255 bytes. Note that due to a bug in Adobe Reader 6 for Windows the & character does not display properly in some info strings.

len (Only for *PDF_set_info2( )*, and only for the C binding) Length of *value* (in bytes) for UTF-16 strings. If *len* = *0* a null-terminated string must be provided.

Details The supplied info value will only be used for the current document, but not for all documents generated within the same *object* scope. If the *autoxmp* option has been supplied to *PDF_begin/end_document( )* PDFlib will automatically create synchronized XMP document metadata from the info entries supplied to *PDF_set_info( )*.

Scope *object, document, page*. If used in *object* scope the supplied values will only be used for the next document.

Table 12.1 Values for the document information field key

key	***explanation***
Subject	*Subject of the document*
Title	*Title of the document*
Creator	*Software used to create the document (as opposed to the Producer of the PDF output, which is always PDFlib). Acrobat will display this entry as »Application«.*
Author	*Author of the document*
Keywords	*Keywords describing the contents of the document*
Trapped	*Indicates whether trapping has been applied to the document. Allowed values are* `True`, `False`, *and* `Unknown`. *In PDF/A mode* `Unknown` *is not allowed.*

1. See dublincore.org

Table 12.1 Values for the document information field key

key	***explanation***
any name other than CreationDate, Producer, and ModDate	*User-defined field. PDFlib supports an arbitrary number of fields. A custom field name should only be supplied once. WIth multiple occurrences of the same field name the last one will be used. See also* `moddate` *option of PDF_begin/end_document( ).*

12.2 XMP Metadata

As an alternative or in addition to document information fields PDFlib supports XMP *(Extensible Metadata Platform*[1]*)* as a framework for specifying metadata. XMP is required, for example, for PDF/A compliance, and is supported by an increasing number of applications. There are several flavors of XMP support in PDFlib as detailed below.

Automatic XMP synchronization for document info keys. If the *autoxmp* option in *PDF_begin/end_document( )* is *true*, PDFlib will synchronize document information fields supplied to *PDF_set_info( )* as well as several internally generated entries (e.g. *CreationDate)* to the corresponding entries in the document-level XMP metadata.

Document info keys which correspond to a well-known element in one of the standard XMP schemas in Table 12.3 will be placed in the corresponding schema. Unknown info keys will be placed in the extended PDF *(pdfx)* schema.

Custom metadata streams. Users can supply full or partial XMP metadata streams to the *metadata* option of various functions. This option expects an XMP stream and will validate it. PDFlib will automatically generate the XDP packet header and trailer.

For document-level metadata PDFlib will add several elements as appropriate (e.g. *CreationDate)*. In PDF/A mode PDFlib will synchronize relevant entries in user-supplied XMP streams to standard document info fields (analogous to *autoxmp* mode which synchronizes document info fields to XMP). However, PDFlib will not synchronize other XMP entries to custom document info fields.

In addition to document-level metadata, XMP can be supplied for pages, fonts, ICC profiles, images, templates, and imported PDF pages. Table 12.2 lists options for XMP metadata.

Table 12.2 Options for XMP metadata in PDF_begin/end_document( ), PDF_begin/end_page_ext( ), PDF_load_font( ), PDF_load_iccprofile( ), PDF_load_image( ), PDF_begin_template_ext( ), PDF_open_pdi_page( )

option	*description*
metadata	*(Option list; PDF 1.4) Supply metadata for the document or another object. The option list may contain the following options:* ***inputencoding*** *(Keyword) The encoding to interpret the supplied data. Default:* `unicode` ***inputformat*** *(Keyword) The format of the supplied data. Default: utf8 (ebcdicutf8 on EBCDIC-based systems), but bytes if inputencoding is an 8-bit encoding* ***filename*** *(Name string; required) The name of a disk-based or virtual file containing well-formed XMP metadata.* *Example:* `metadata={filename=info.xmp inputencoding=winansi}`

Standard schemas and namespace URIs. PDFlib internally knows about the XMP schemas listed in Table 12.3, and will use the namespace URIs and prefixes listed in the table (see XMP reference for details on these schemas). Custom schemas in user-supplied XMP must not use any of the namespace URIs in Table 12.3. Similarly, two custom schemas must not use the same namespace URI.

1. See www.adobe.com/products/xmp

Table 12.3 XMP schemas known to PDFlib internally

Schema name and description	namespace URI	namespace prefix
Dublin core (common document properties)	http://purl.org/dc/elements/1.1/	dc
XMP basic schema	http://ns.adobe.com/xap/1.0/	xmp
XMP rights management schema	http://ns.adobe.com/xap/1.0/rights/	xmpRights
XMP media management schema	http://ns.adobe.com/xap/1.0/mm/	xmpMM
XMP basic job ticket schema	http://ns.adobe.com/xap/1.0/bj	xmpBJ
XMP paged-text schema	http://ns.adobe.com/xap/1.0/t/pg/	xmpTPg
XMP dynamic media schema	http://ns.adobe.com/xmp/1.0/DynamicMedia/	xmpDM
Adobe PDF schema	http://ns.adobe.com/pdf/1.3/	pdf
PDF/A identification schema	http://www.aiim.org/pdfa/ns/id/[1]	pdfaid
extended PDF schema (not PDF/X!)	http://ns.adobe.com/pdfx/1.3/	pdfx
Photoshop schema	http://ns.adobe.com/photoshop/1.0/	photoshop
Camera raw schema	http://ns.adobe.com/camera-rawsettings/1.0/	crs
EXIF schema for TIFF properties	http://ns.adobe.com/tiff/1.0/	tiff
EXIF schema for EXIF-specific properties	http://ns.adobe.com/exif/1.0/	exif
EXIF schema for additional EXIF properties	http://ns.adobe.com/exif/1.0/aux/	aux
IPTC core schema	http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/	Iptc4xmpCore

1. In the first publication of the PDF/A standard (ISO 19005-1) the PDF/A namespace URI was incorrectly stated as http://www.aiim.org/pdfa/ns/id (the trailing slash was missing). This has been corrected in the 2006 corrigendum of the standard. Acrobat 7.07 incorrectly creates and expects http://www.aiim.org/pdfa/ns/id.html, while Acrobat 8 works with the corrected value.

Example with custom XMP. The following example shows XMP with a custom schema, as it could be supplied to the *metadata* option:

```
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
 xmlns:iX='http://ns.adobe.com/iX/1.0/'>
 <rdf:Description rdf:about='' xmlns:xap='http://ns.adobe.com/xap/1.0/'>
  <xap:CreatorTool>ACME Reporting Tool 1.2.3</xap:CreatorTool>
  <xap:Identifier>
   <rdf:Bag>
    <rdf:li>12345</rdf:li>
   </rdf:Bag>
  </xap:Identifier>
 </rdf:Description>

 <rdf:Description rdf:about='' xmlns:acmedoc='http://ns.acme.com/acmedoc/1.0/'>
  <acmedoc:CLIENT>299</acmedoc:CLIENT>
  <acmedoc:DESTINATION>XYZ</acmedoc:DESTINATION>
  <acmedoc:OBJ>ACCOUNT</acmedoc:OBJ>
  <acmedoc:ARCHIVE>KDGTCVJEU</acmedoc:ARCHIVE>
  <acmedoc:ID>00709003280044000492O8</acmedoc:ID>
 </rdf:Description>
</rdf:RDF>
```

12.3 Tagged PDF

The *tagged* option in *PDF_begin_document( )* must have been set to *true* in order to generate Tagged PDF. The *lang* option must be provided as well.

Tagged PDF mode will automatically be activated if the *pdfa* option in *PDF_begin_document( )* has been set to *PDF/A-1a:2005*.

C++ Java ***int begin_item(String tag, String optlist)***

Perl PHP ***int PDF_begin_item(resource p, string tag, string optlist)***

C ***int PDF_begin_item(PDF *p, const char *tag, const char *optlist)***

Open a structure element or other content item with attributes supplied as options.

tag The item's element type according to Table 12.4. It must be one of the standard structure types allowed for the current PDF compatibility level, or a pseudo tag.

Table 12.4 Standard item tags

category	***tags***
grouping	Document, Part, Art, Sect, Div, BlockQuote, Caption, TOC, TOCI, Index, NonStruct, Private
paragraph-like	P, H, H1-H6 *(BLSEs)*
list	L, LI, Lbl, LBody *(BLSEs)*
table	Table *(BLSE)*, TR, TH, TD, THead[1], TBody[1], TFoot[1]
inline-level	Span, TagSuspect[2], Quote, Note, Reference, BibEntry, Code, *(ILSEs)*
illustration	Figure, Formula, Form
Japanese	Ruby[1] *(grouping)*, RB[1], RT[1], RP[1], Warichu[1] *(grouping)*, WT[1], WP[1]
pseudo tags	*The following tags create items which are not structure elements:* ***Artifact*** *Specifies an artifact, to be distinguished from real page content.* ***ASpan*** *(Accessibility span; will be written to PDF as* Span*, but must be distinguished from the inline-level item* Span*) Can be used to attach accessibility properties to content which does not belong to a structure element, or which resembles only a fraction of a structure element.* ***ReversedChars*** *Specifies text in a right-to-left language with reversed character sequence. This is useful for making Hebrew or Arabic text searchable in Acrobat.* ***Clip*** *Specifies a marked clipping sequence. This is a sequence containing only clipping paths or text in rendering mode 7, but no visible graphics or PDF_save( ) / PDF_restore( ).*

1. Requires PDF 1.5 or above
2. Requires PDF 1.6 or above

optlist An option list specifying details of the item according to Table 12.5. All inheritable settings will be inherited to child elements, and therefore need not be repeated. All properties of an item must be set here since they cannot be modified later. The following options can be used:
ActualText, Alt, artifacttype, Attached, BBox, ColSpan, E, hypertextencoding, index, inline, Lang, parent, RowSpan, Scope, Title

Returns An item handle which can be used in subsequent item-related calls.

Details This function generates the document's structure tree, which is essential for Tagged PDF. The position of the new element in the structure tree can be controlled with the *parent* and *index* options. Structure elements can be nested to an arbitrary level. Regular items are not bound to the page where they have been opened, but can be continued on an arbitrary number of pages.

Scope *page* for inline items, and for regular items also *document;* must always be paired with a matching *PDF_end_item( )* call. This function is only allowed in Tagged PDF mode.

Table 12.5 Options for the properties of structure and pseudo tags with PDF_begin_item( )

option	explanation
ActualText	*(Hypertext string; not for pseudo tags except in PDF 1.5 with* ASpan*; not for* TagSuspect*; required for text in fonts which are not Unicode-compatible) Equivalent replacement text for the content item. It should be provided for text content which is represented in some non-standard way, such as ligatures, swash characters in illustrations, drop caps, etc. If this option is used in PDF 1.4 mode the* inline *option must be set to false.*
Alt	*(Hypertext string; not for pseudo tags except in PDF 1.5 with* ASpan*; not for* TagSuspect*) Alternate description for the content item. It should be provided for figures, images, etc., which cannot be recognized as text. Alternate text for images is required for accessibility. If this option is used in PDF 1.4 mode the* inline *option must be set to* false*.*
artifacttype	*(Keyword; only for* tag=Artifact*) Identifies the artifact type of the content item:* Pagination*,* Layout*, or* Page
Attached	*(Keyword list; only for* tag=Artifact *and* artifacttype=Pagination*) A list containing one to four of the keywords* Top*,* Bottom*,* Left*, and* Right
BBox	*(Rectangle; only for* tag=Artifact *as well as all table and illustration tags; optional, but recommended for reflow) The artifact's bounding box in the default coordinate system (if* usercoordinates= false*) or the user coordinate system (if* usercoordinates=true*). If this option has not been supplied PDFlib will automatically create a* BBox *entry for imported images and PDF pages.*
ColSpan	*(Integer; only for* tag=TH *and* TD*) Number of table columns spanned by a cell.*
E	*(Hypertext string; not for pseudo tags except* ASpan*; not for* TagSuspect*; requires PDF 1.5 for structure tags) Abbreviation expansion for the content item. It should be provided for explaining abbreviations and acronyms. Acrobat's Read Aloud feature will consider the expansion text as a separate word even in the absence of explicit word breaks.*
hypertext-encoding	*(Keyword) Specifies the encoding for the supplied text. An empty string is equivalent to unicode. Default: empty string for Unicode-capable language bindings, otherwise auto.*
index	*(Integer; not for pseudo tags and* TagSuspect*) The index at which to insert the element within the parent. Values between 0 and the current number of children will be used to insert the item at that specific location within the parent. The value -1 can be used to insert the element as the last item. Default: -1*
inline	*(Boolean; only for* tag=ASpan *and all inline-level tags except* TagSuspect*) If* true*, the content item will be written inline, and no structure element will be created. Default:* true
Lang	*(String; not for* TagSuspect *and pseudo tags except* ASpan*) Language identifier for the content item in the format described in Table 2.3 for the* lang *option. This can be used to override the document's dominant language for individual content items.*
parent	*(Item handle; not for* TagSuspect *and pseudo tags) The item handle of the element's parent, as returned by another call to PDF_begin_item( ). The value 0 refers to the structure tree root. -1 refers to the parent of the least recently opened element on the current page. In other words,* parent=-1 *opens a sibling of the current element. Default: -1*
RowSpan	*(Integer; only for* tag=TH *and* TD*) The number of table rows spanned by a cell.*

Table 12.5 Options for the properties of structure and pseudo tags with PDF_begin_item( )

option	explanation
Scope	*(Keyword; only for* `tag=TH`*; PDF 1.5 or above) One of the keywords* `Row`*,* `Column`*, or* `Both` *indicating whether the header cell applies to the rest of the cells in the row that contains it, the column that contains it, or both the row and the column that contain it.*
Title	*(Hypertext string; not for inline and pseudo tags) Name of the structure element*

C++ Java ***void end_item(int id)***
Perl PHP ***PDF_end_item(resource p, int id)***
C ***void PDF_end_item(PDF *p, int id)***

Close a structure element or other content item.

id The item's handle, which must have been retrieved with *PDF_begin_item( )*.

Details All inline items must be closed before the end of the page. All regular items must be closed before the end of the document. However, it is strongly recommended to close all regular items as soon as they are completed to reduce memory consumption. An item can only be closed if all of its children have been closed before. After closing an item its parent will become the active item.

Scope *page* for inline items, and for regular items also *document;* must always be paired with a matching *PDF_begin_item( )* call. This function is only allowed in Tagged PDF mode.

C++ Java ***void activate_item(int id)***
Perl PHP ***PDF_activate_item(resource p, int id)***
C ***void PDF_activate_item(PDF *p, int id)***

Activate a previously created structure element or other content item.

id The item's handle, which must have been retrieved with *PDF_begin_item( ),* and must not yet have been closed. Pseudo and inline-level items can not be activated.

Details Putting aside a structure element and activating it later gives additional flexibility for efficiently creating Tagged PDF pages even when there are multiple parallel structure branches on a page, e.g. with multi-column layouts or text inserts which interrupt the main text.

Scope *document, page;* This function is only allowed in Tagged PDF mode.

A List of all Functions

Click on a function name to jump to the corresponding description.

General Functions	***Text Output Functions***	***Graphics Functions***	***PDI and pCOS Functions***
PDF_get_value	*PDF_set_text_pos*	*PDF_setdash*	*PDF_open_pdi_document*
PDF_set_value	*PDF_show*	*PDF_setdashpattern*	*PDF_open_pdi_callback*
PDF_get_parameter	*PDF_xshow*	*PDF_setflat*	*PDF_close_pdi_document*
PDF_set_parameter	*PDF_show_xy*	*PDF_setlinejoin*	*PDF_open_pdi_page*
PDF_new	*PDF_continue_text*	*PDF_setlinecap*	*PDF_close_pdi_page*
PDF_new2	*PDF_stringwidth*	*PDF_setmiterlimit*	*PDF_fit_pdi_page*
PDF_delete		*PDF_setlinewidth*	*PDF_process_pdi*
PDF_begin_document	***Text Formatting Functions***	*PDF_initgraphics*	*PDF_pcos_get_number*
PDF_begin_document_callback	*PDF_fit_textline*	*PDF_save*	*PDF_pcos_get_string*
PDF_end_document	*PDF_info_textline*	*PDF_restore*	*PDF_pcos_get_stream*
PDF_get_buffer	*PDF_add_textflow*	*PDF_translate*	
PDF_begin_page_ext	*PDF_create_textflow*	*PDF_scale*	***Block Filling Functions (PPS)***
PDF_end_page_ext	*PDF_fit_textflow*	*PDF_rotate*	*PDF_fill_textblock*
PDF_suspend_page	*PDF_info_textflow*	*PDF_skew*	*PDF_fill_imageblock*
PDF_resume_page	*PDF_delete_textflow*	*PDF_concat*	*PDF_fill_pdfblock*
PDF_create_pvf		*PDF_setmatrix*	
PDF_delete_pvf	***Table Formatting Functions***	*PDF_create_gstate*	***Interactive Features***
PDF_get_errnum	*PDF_add_table_cell*	*PDF_set_gstate*	*PDF_create_action*
PDF_get_errmsg	*PDF_fit_table*	*PDF_moveto*	*PDF_add_nameddest*
PDF_get_apiname	*PDF_info_table*	*PDF_lineto*	*PDF_create_annotation*
PDF_get_opaque	*PDF_delete_table*	*PDF_curveto*	*PDF_create_field*
	PDF_info_matchbox	*PDF_circle*	*PDF_create_fieldgroup*
Font Functions		*PDF_arc*	*PDF_create_bookmark*
PDF_load_font	***Color Functions***	*PDF_arcn*	
PDF_setfont	*PDF_setcolor*	*PDF_rect*	***Multimedia***
PDF_info_font	*PDF_makespotcolor*	*PDF_closepath*	*PDF_load_3ddata*
PDF_begin_font	*PDF_load_iccprofile*	*PDF_stroke*	*PDF_create_3dview*
PDF_end_font	*PDF_begin_pattern*	*PDF_closepath_stroke*	
PDF_begin_glyph	*PDF_end_pattern*	*PDF_fill*	***Document Interchange***
PDF_end_glyph	*PDF_shading_pattern*	*PDF_fill_stroke*	*PDF_set_info*
PDF_encoding_set_char	*PDF_shfill*	*PDF_closepath_fill_stroke*	*PDF_begin_item*
PDF_utf16_to_utf8	*PDF_shading*	*PDF_clip*	*PDF_end_item*
PDF_utf8_to_utf16		*PDF_endpath*	*PDF_activate_item*
PDF_utf32_to_utf16	***Image Functions***	*PDF_define_layer*	
	PDF_load_image	*PDF_set_layer_dependency*	
	PDF_close_image	*PDF_begin_layer*	
	PDF_fit_image	*PDF_end_layer*	
	PDF_begin_template_ext		
	PDF_end_template		
	PDF_add_thumbnail		

B List of all Parameters

This section lists all keywords for *PDF_get/set_parameter( )* and *PDF_get/set_value( )*. Click on a keyword to jump to the corresponding description.

category	PDF_get/set_parameter( )	PDF_get/set_value( )
setup	*license*[1], *licensefile*, *nodemostamp*, *resourcefile*, *scope*[1], *SearchPath*, *string*[1], *asciifile*, *errorpolicy*	*compress*
logging	*logging*[1], *logmsg*[1]	
versioning	*version*[1]	*major*, *minor*, *revision*[2]
page	*topdown*	*pagewidth*, *pageheight*
font handling	*Encoding*, *FontAFM*, *FontPFM*, *FontOutline*, *HostFont*	*font*[2], *fontsize*[2]
simple text output	*autospace*, *charref*, *escapesequence*, *fakebold*, *glyphcheck*, *kerning*, *textformat*, *underline*, *overline*, *strikeout*	*charspacing*, *horizscaling*, *italicangle*, *leading*, *textrendering*, *textrise*, *textx*[2], *texty*[2], *underlineposition*, *underlinewidth*, *wordspacing*
graphics	*fillrule*	*currentx*[2], *currenty*[2], *ctm_a*[2], *ctm_b*[2], *ctm_c*[2], *ctm_d*[2], *ctm_e*[2], *ctm_f*[2]
color	*preserveoldpantonenames*, *spotcolorlookup*	
ICC profiles	*ICCProfile*, *StandardOutputIntent*	*icccomponents*[2], *setcolor:iccprofilegray*, *setcolor:iccprofilergb*, *setcolor:iccprofilecmyk*
image	*honoriccprofile*, *renderingintent*	*imagewidth*[2], *imageheight*[2], *image:iccprofile*[2], *orientation*[2], *resx*[2], *resy*[2]
PDI	*pdi*[1]	
interactive	*hypertextencoding*, *hypertextformat*, *usehypertextencoding*, *usercoordinates*	

1. Only for PDF_get_parameter( )
2. Only for PDF_get_value( )

C List of all Options

This index contains an alphabetical list of all options along with the functions in which they can be used. Click on an option name to jump to the corresponding description.

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

D Revision History

Date	Changes
October 03, 2006	▸ Updates and restructuring for PDFlib 7.0.0; split the manual in tutorial and API reference
February 21, 2006	▸ Various updates and corrections for PDFlib 6.0.3; added Ruby section
August 09, 2005	▸ Various updates and corrections for PDFlib 6.0.2
November 17, 2004	▸ Minor updates and corrections for PDFlib 6.0.1 ▸ introduced new format for language-specific function prototypes in chapter 8 ▸ added hypertext examples in chapter 3
June 18, 2004	▸ Major changes for PDFlib 6
January 21, 2004	▸ Minor additions and corrections for PDFlib 5.0.3
September 15, 2003	▸ Minor additions and corrections for PDFlib 5.0.2; added block specification
May 26, 2003	▸ Minor updates and corrections for PDFlib 5.0.1
March 26, 2003	▸ Major changes and rewrite for PDFlib 5.0.0
June 14, 2002	▸ Minor changes for PDFlib 4.0.3 and extensions for the .NET binding
January 26, 2002	▸ Minor changes for PDFlib 4.0.2 and extensions for the IBM eServer edition
May 17, 2001	▸ Minor changes for PDFlib 4.0.1
April 1, 2001	▸ Documents PDI and other features of PDFlib 4.0.0
February 5, 2001	▸ Documents the template and CMYK features in PDFlib 3.5.0
December 22, 2000	▸ ColdFusion documentation and additions for PDFlib 3.03; separate COM edition of the manual
August 8, 2000	▸ Delphi documentation and minor additions for PDFlib 3.02
July 1, 2000	▸ Additions and clarifications for PDFlib 3.01
Feb. 20, 2000	▸ Changes for PDFlib 3.0
Aug. 2, 1999	▸ Minor changes and additions for PDFlib 2.01
June 29, 1999	▸ Separate sections for the individual language bindings ▸ Extensions for PDFlib 2.0
Feb. 1, 1999	▸ Minor changes for PDFlib 1.0 (not publicly released)
Aug. 10, 1998	▸ Extensions for PDFlib 0.7 (only for a single customer)
July 8, 1998	▸ First attempt at describing PDFlib scripting support in PDFlib 0.6
Feb. 25, 1998	▸ Slightly expanded the manual to cover PDFlib 0.5
Sept. 22, 1997	▸ First public release of PDFlib 0.4 and this manual

Index

Note that functions, parameters, and options are listed in separate appendices.

R

S

T

U

W

X