

# PDFlib, PDFlib+PDI, PPS

A library for generating PDF on the fly  
Version 8.0.6

## API Reference

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



Copyright © 1997–2013 PDFlib GmbH and Thomas Merz. All rights reserved.  
PDFlib users are granted permission to reproduce printed or digital copies of this manual for internal use.

PDFlib GmbH  
Franziska-Bilek-Weg 9, 80339 München, Germany  
www.pdflib.com  
phone +49 • 89 • 452 33 84-0  
fax +49 • 89 • 452 33 84-99

If you have questions check the PDFlib mailing list and archive at [tech.groups.yahoo.com/group/pdflib](http://tech.groups.yahoo.com/group/pdflib)

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

This publication and the information herein is furnished as is, is subject to change without notice, and should not be construed as a commitment by PDFlib GmbH. PDFlib GmbH assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes and noninfringement of third party rights.

PDFlib and the PDFlib logo are registered trademarks of PDFlib GmbH. PDFlib licensees are granted the right to use the PDFlib name and logo in their product documentation. However, this is not required.

Adobe, Acrobat, PostScript, and XMP are trademarks of Adobe Systems Inc. AIX, IBM, OS/390, WebSphere, iSeries, and zSeries are trademarks of International Business Machines Corporation. ActiveX, Microsoft, OpenType, and Windows are trademarks of Microsoft Corporation. Apple, Macintosh and TrueType are trademarks of Apple Computer, Inc. Unicode and the Unicode logo are trademarks of Unicode, Inc. Unix is a trademark of The Open Group. Java and Solaris are trademarks of Sun Microsystems, Inc. HKS is a registered trademark of the HKS brand association: Hostmann-Steinberg, K+E Printing Inks, Schmincke. Other company product and service names may be trademarks or service marks of others.

PANTONE® colors displayed in the software application or in the user documentation may not match PANTONE-identified standards. Consult current PANTONE Color Publications for accurate color. PANTONE® and other Pantone, Inc. trademarks are the property of Pantone, Inc. © Pantone, Inc., 2003. Pantone, Inc. is the copyright owner of color data and/or software which are licensed to PDFlib GmbH to distribute for use only in combination with PDFlib Software. PANTONE Color Data and/or Software shall not be copied onto another disk or into memory unless as part of the execution of PDFlib Software.

PDFlib contains modified parts of the following third-party software:

ICCLib, Copyright © 1997-2002 Graeme W. Gill  
GIF image decoder, Copyright © 1990-1994 David Koblas  
PNG image reference library (libpng), Copyright © 1998-2004 Glenn Randers-Pehrson  
Zlib compression library, Copyright © 1995-2002 Jean-loup Gailly and Mark Adler  
TIFFlib image library, Copyright © 1988-1997 Sam Leffler, Copyright © 1991-1997 Silicon Graphics, Inc.  
Cryptographic software written by Eric Young, Copyright © 1995-1998 Eric Young ([eay@cryptsoft.com](mailto:eay@cryptsoft.com))  
Independent JPEG Group's JPEG software, Copyright © 1991-1998, Thomas G. Lane  
Cryptographic software, Copyright © 1998-2002 The OpenSSL Project ([www.openssl.org](http://www.openssl.org))  
Expat XML parser, Copyright © 1998, 1999, 2000 Thai Open Source Software Center Ltd  
ICU International Components for Unicode, Copyright © 1995-2009 International Business Machines Corporation and others  
Reference sRGB ICC color profile data, Copyright (c) 1998 Hewlett-Packard Company

PDFlib contains the RSA Security, Inc. MD5 message digest algorithm.



# Contents

## 1 Option Lists 7

- 1.1 Option List Syntax 7
- 1.2 Basic Types 10
- 1.3 Fontsize, Color, and Action 12
- 1.4 Geometric Types 14
- 1.5 Limits 16

## 2 General Functions 17

- 2.1 Function Scopes 17
- 2.2 Parameter and Option Handling 19
- 2.3 Setup 22
- 2.4 PDFlib Virtual File System (PVF) 25
- 2.5 Exception Handling 27
- 2.6 Logging 29

## 3 Document and Page Functions 31

- 3.1 Document Functions 31
- 3.2 Fetching PDF Documents from Memory 39
- 3.3 Page Functions 40
- 3.4 Layers 45

## 4 Font and Text Functions 51

- 4.1 Font Handling 51
- 4.2 Type 3 Font Definition 62
- 4.3 Encoding Definition 65
- 4.4 Simple Text Output 66
- 4.5 Unicode Conversion Functions 71

## 5 Text and Table Formatting 75

- 5.1 Text Options 75
- 5.2 Single-Line Text with Textlines 79
- 5.3 Multi-Line Text with Textflows 83
- 5.4 Table Formatting 99

## 6 Object Fitting and Matchboxes 109

- 6.1 Object Fitting 109
- 6.2 Matchboxes 115

<b>7</b>	<b>Graphics Functions</b>	<b>119</b>
7.1	Graphics Appearance Parameters and Options	119
7.2	Graphics State	122
7.3	Coordinate System Transformations	127
7.4	Path Construction	130
7.5	Painting and Clipping	134
7.6	Path Objects	136
<b>8</b>	<b>Color Functions</b>	<b>141</b>
8.1	Setting Color and Color Space	141
8.2	ICC Profiles	145
8.3	Patterns and Shadings	149
<b>9</b>	<b>Image and Template Functions</b>	<b>153</b>
9.1	Images	154
9.2	Templates	162
9.3	Thumbnails	165
<b>10</b>	<b>PDF Import (PDI) and pCOS Functions</b>	<b>167</b>
10.1	Document Functions	167
10.2	Page Functions	171
10.3	Other PDI Processing	176
10.4	pCOS Functions	177
<b>11</b>	<b>Block Filling Functions (PPS)</b>	<b>181</b>
11.1	Rectangle Options for Block Filling Functions	181
11.2	Textline and Textflow Blocks	182
11.3	Image Blocks	184
11.4	PDF Blocks	185
<b>12</b>	<b>Interactive Features</b>	<b>187</b>
12.1	Parameters for Interactive Elements	187
12.2	Actions	187
12.3	Named Destinations	192
12.4	Annotations	194
12.5	Form Fields	202
12.6	Bookmarks	209
12.7	PDF Packages and Portfolios	211

## **13 3D and Geospatial Features 217**

13.1 3D Artwork 217

13.2 Geospatial Features 222

## **14 Document Interchange 225**

14.1 Document Information Fields 225

14.2 XMP Metadata 227

14.3 Tagged PDF 228

14.4 Marked Content 232

**A List of all Functions 235**

**B List of all Parameters 237**

**C List of all Options and Keywords 239**

**D Revision History 253**

**Index 255**



# 1 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 can contain an arbitrary number of options. Option lists support various data types and composite data like lists. In most language bindings optlists can easily be constructed by concatenating the required keywords and values.

*Bindings* C language binding: you may want to use the *sprintf()* function for constructing optlists.

## 1.1 Option List Syntax

**Formal option list syntax definition.** Option lists must be constructed according to following rules:

- ▶ All elements (keys and values) in an option list must be separated by one or more of the following separator characters: space, tab, carriage return, newline, equal sign '='.
- ▶ An outermost pair of enclosing braces is not part of the element. The sequence `{ }` designates an empty element.
- ▶ Separators within the outermost pair of braces no longer split elements, but are part of the element. Therefore, an element which contains separators must be enclosed with braces.
- ▶ If an element contains brace characters these must be protected with a preceding backslash character.
- ▶ If an element contains a sequence of one or more backslash characters in front of a brace, each backslash in the sequence must be protected with another backslash character.
- ▶ Option lists must not contain binary zero values.

An option may have a list value according to its documentation in this PDFlib Reference. List values contain one or more elements (which may themselves be lists). They are separated according to the rules above, with the only difference that the equal sign is no longer treated as a separator.

*Note* Option names (i.e. the key) never contain hyphen characters. Keep this in mind since the tables with option descriptions may sometimes contain long option names which are hyphenated. The hyphen must be omitted when supplying the option in an option list.

**Simple option lists.** In many cases option lists will contain one or more key/value pairs. Keys and values, as well as multiple key/value pairs must be separated by one or more whitespace characters (space, tab, carriage return, newline). Alternatively, keys can be separated from values by an equal sign '=':

```
key=value  
key = value  
key value  
key1 = value1 key2 = value2
```

To increase readability we recommend to use equal signs between key and value and whitespace between adjacent key/value pairs.

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 *value2* after processing the option list:

```
key=value1 key=value2
```

**List values.** Lists contain one or more separated values, which may be simple values or list values in turn. Lists are bracketed with { and } braces, and the values in the list must be separated by whitespace characters. Examples:

```
dasharray={11 22 33}           (list containing three numbers)
position={ center bottom }     (list containing two keywords)
```

A list may also contain nested lists. In this case the lists must also be separated from each other by whitespace. While a separator must be inserted between adjacent } and { characters, it can be omitted between braces of the same kind:

```
polylinelist={{10 20 30 40} {50 60 70 80}} (list containing two lists)
```

If the list contains exactly one list the braces for the nested list must not be omitted:

```
polylinelist={{10 20 30 40}} (list containing one nested list)
```

**Nested option lists and list values.** Some options 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. Several examples are listed below.

The value of the option *metadata* is an option list which itself contains a single option *filename*:

```
metadata={filename=info.xmp}
```

The value of the option *fill* is a list of option lists containing a single option list:

```
fill={{ area=table fillcolor={rgb 1 0 0} }}
```

The value of the option *fill* is a list of option lists containing two option lists:

```
fill={{ area=rowodd fillcolor={rgb 0 1 0} } { area=roweven fillcolor={rgb 1 0 0} }}
```

List containing one option list with a value that includes spaces:

```
attachments={{filename={foo bar.xml} }}
```

List containing three strings:

```
itemnamelist = { {Isaac Newton} {James Clark Maxwell} {Albert Einstein} }
```

List containing two keywords:

```
position={left bottom}
```

List containing different types (float and keyword):



# 1.2 Basic Types

**String.** Strings are plain ASCII strings (or EBCDIC strings on EBCDIC platforms) which are generally used for non-localizable keywords. Strings containing whitespace or '=' characters must be bracketed with { and }:

```
password={ secret string }           (string value contains three blanks)
contents={length=3mm}                (string value containing one equal sign)
```

The characters { and } must be preceded by an additional \ character if they are supposed to be part of the string:

```
password={weird\}string}            (string value contains a right brace)
```

A backslash in front of the closing brace of an element must be preceded by a backslash character:

```
filename={C:\path\name\}           (string ends with a single backslash)
```

An empty string can be constructed with a pair of braces:

```
{}
```

Content strings, hypertext strings and name strings: these can hold Unicode content in various formats. Single bytes can be expressed by an escape sequence if the parameter *escapesequence* is set. For details on these string types and encoding choices for string options see the *PDFlib Tutorial*.

Non-Unicode capable language bindings: if an option list starts with a [EBCDIC-]UTF-8 BOM, each content, hypertext or name string of the option list will be interpreted as a [EBCDIC-]UTF-8 string.

**Unichar.** A Unichar is a single Unicode value where several syntax variants are supported: decimal values ≥ 10 (e.g. 173), hexadecimal values prefixed with x, X, 0x, 0X, 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. Examples:

```
replacementchar=?                    (literal)
replacementchar=63                    (decimal)
replacementchar=x3F                   (hexadecimal)
replacementchar=0x3F                  (hexadecimal)
replacementchar=U+003F                (Unicode notation)
replacementchar=euro                  (HTML character reference)
replacementchar=.question             (standard glyph name reference)
replacementchar=.marina                (font-specific glyph name reference)
```

Single characters which happen to be a number are treated literally, not as decimal Unicode values:

```
replacementchar=3                     (U+0033 THREE, not U+0003!)
```

Unichars must be in the hexadecimal range 0-0x10FFFF (decimal 0-1114111). However, some options are restricted to the range 0-0xFFFF (0-65535). This is noted in the respective option description.

**Unicode range.** A Unicode range identifies a contiguous range of Unicode characters via start and end characters of the range. The start and end values of a Unicode range must be separated by a minus sign '-' without any spaces, e.g.

```
forcechars={U+03AC-U+03CE}
```

**Boolean.** Booleans have the values *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*:

```
embedding           (equivalent to embedding=true)
noembedding         (equivalent to embedding=false)
```

**Keyword.** An option of type keyword can hold one of a predefined list of fixed keywords. Example:

```
blendmode=overlay
```

For some options the value hold either a number or a keyword.

**Number.** Option list support several numerical types.

Integer types can hold decimal and hexadecimal integers. Positive integers starting with x, X, ox, or oX specify hexadecimal values:

```
-12345
0
0xFF
```

Floats can hold decimal floating point or integer numbers; period and comma can be used as decimal separators for floating point values. Exponential notation is also supported. The following values are all equivalent:

```
size = -123.45
size = -123,45
size = -1.2345E2
size = -1.2345e+2
```

Percentages are numbers with a % character directly after the numerical value. Some options allow negative percentages:

```
leading=120%
topoffset=-20.5%
```

**Handle.** Handles identify various types of objects, e.g. fonts, images, or actions. Technically these are integer values which have been returned earlier by an API function. For example, a font handle is returned by *PDF\_load\_font()*. Handles must always be treated as opaque types; they must never be modified or created by the application directly (as opposed to using a handle returned by an API function). Handles must always be valid for the respective type of object. For example, an option which expects an image handle must be supplied with a font handle, although both handles are integer types.

## 1.3 Fontsize, Color, and Action

**Fontsize.** A fontsize can be defined in several ways which allow the size of text to be specified in absolute values, relative to some external entity, or relative to some font property. In general the fontsize must be different from 0 unless the option description mentions otherwise.

In the most common case a fontsize contains a single float value which specifies refers to units in the user coordinate system:

```
fontsize = 12
```

The second variant contains a percentage, where the basis of the percentage depends on the context (e.g. the width of the fitbox for `PDF_fit_textline()`):

```
fontsize = 8%
```

In the third variant, the fontsize is specified as an option list which must contain a keyword and a number. The keyword describes the desired font metric according to Table 1.1, and the number contains the desired size. PDFlib will calculate the proper fontsize so that the selected text metric matches the supplied value:

```
fontsize = {capheight 5}
```

Table 1.1 Suboptions for options of type fontsize

<i>option</i>	<i>explanation</i>
<i>ascender</i>	<i>The number will be interpreted as ascender height.</i>
<i>bodyheight</i>	<i>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.</i>
<i>capheight</i>	<i>The number will be interpreted as capital letter height.</i>
<i>xheight</i>	<i>The number will be interpreted as lowercase letter height.</i>

**Color.** Colors can be defined in three different forms: using an RGB color name, hexadecimal RGB values, or a flexible option list for colors in any color space.

In the first form all valid color names from SVG 1.1 can be supplied directly to specify an RGB color, e.g.

```
strokecolor=pink
```

The color names are case-insensitive. A list of valid color names can be found at the following location:

```
www.w3.org/TR/SVG11/types.html#ColorKeywords
```

In the second form a hash '#' character followed by any combination of three pairs of hexadecimal digits `00-FF` can be supplied to specify an RGB color value, e.g.

```
strokecolor=#FFC0CB
```

In the third form an color option list specified a color space and color value. A color option list contains a colorspace 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 8.1, «Setting Color and Color Space», page 141). Table 1.2 contains specific descriptions and examples. As detailed in the respective function descriptions, a particular option list may supply only a subset of the keywords presented above.

*Cookbook* A full code sample can be found in the *Cookbook* topic `color/starter_color`.

Table 1.2 Keywords for the color data type in option lists

keyword	additional values	example
<b>gray</b>	single float value for the grayscale color space	{ gray 0.5 }
<b>rgb</b>	three float values for the RGB color space	{ rgb 1 0 0 }
<b>(no keyword)</b>	HTML color name or hexadecimal values for an RGB color	pink #FFC0CB
<b>cmymk</b>	four float values for the CMYK color space	{ cmyk 0 1 0 0 }
<b>lab</b>	three float values for the Lab color space	{ lab 100 50 30 }
<b>spot</b>	spot color handle and a float specifying the tint value	{ spot 1 0.8 }
<b>spotname</b>	(up to 63 bytes; fewer Unicode characters depending on format and encoding) spot color name and a float specifying the tint value	{ spotname {PANTONE 281 U} 0.5 }
<b>spotname</b>	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 } }
<b>iccbasedgray</b>	single float value	{ iccbasedgray 0.5 }
<b>iccbasedrgb</b>	three float values	{ iccbasedrgb 1 0 0 }
<b>iccbasedcmyk</b>	four float values	{ iccbasedcmyk 0 1 0 0 }
<b>pattern</b>	pattern handle	{ pattern 1 }
<b>none</b>	specifies the absence of color	none

**Action list.** 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.

List containing a single trigger with three actions:

```
action={ activate { 0 1 2 } }
```

List containing three triggers with one action for each:

```
action={ keystroke=0 format=1 validate=2 }
```

## 1.4 Geometric Types

**Line.** A line is a list of four float values specifying the  $x$  and  $y$  coordinates of the start and end point of a line segment. The coordinate system for interpreting the coordinates (default or user coordinate system) varies depending on the option, and is documented separately:

```
line = {10 40 130 90}
```

**Polyline.** A polyline is a list containing an even number  $n$  of float values with  $n > 2$ . Each pair in the list specifies the  $x$  and  $y$  coordinates of a point; these points will be connected by line segments. The coordinate system for interpreting the coordinates (default or user coordinate system) varies depending on the option, and is documented separately:

```
polyline = {10 20 30 40 50 60}
```

The following option lists are equivalent:

```
polyline = {10 20 30r 40r 50r 60r}  
polyline = {10 20 40 60 90 120}
```

Quadrilaterals are a special type of polylines: these are rectangles which may be rotated and for which exactly four points must be specified.

Another special type are polygons: these are polylines which will automatically be closed by a line segment.

**Rectangle.** A rectangle is a list of four float values specifying the  $x$  and  $y$  coordinates of the lower left and upper right corners of a rectangle. The coordinate system for interpreting the coordinates (default or user coordinate system) varies depending on the option, and is documented separately. Some options accept percentages, where the basis for the percentage depends on the context (e.g. the fitbox of a Textflow). Relative coordinates can be supplied by adding the suffix  $r$  immediately after a number. Within a coordinate list a relative coordinate relates to the previous  $x$  or  $y$  coordinate. Relative coordinates at the beginning of a list relate to the origin, i.e. they are absolute coordinates.

Examples:

```
cropbox={ 0 0 500 600 }  
box={40% 30% 50% 70%}
```

The following options are equivalent:

```
box={12 34 56r 78r}  
box={12 34 68 112}
```

**Circle.** A circle is specified as a list of four float values where the first pair specifies the  $x$  and  $y$  coordinates of the center, and the second pair specifies the  $x$  and  $y$  coordinates of an arbitrary point on the circle. The coordinate system for interpreting the coordinates (default or user coordinate system) varies depending on the option, and is documented separately:

```
circle={200 325 200 200}
```

**Curve list.** A curve list consists of two or more connected third-order Bézier curve segments. A Bézier curve is specified by four control points. The first control point is the starting point and the fourth point is the end point of the curve. The second and third point control the shape of the curve. In a curve list the last point of a segment serves as the first point for the next segment. A curve list is therefore specified as a list of  $6 \times n$  float values with  $n \geq 2$ :

```
curve={200 700 240 600 80 580 400 660 400 660 440 620}
```

The last control point will become the new current point after drawing the curves.

## 1.5 Limits

PDFlib imposes limits on certain entities in order to create PDF output which conforms to the limitations imposed by the PDF Reference, Acrobat, or some PDF standard. These limits are documented below.

The following limits will be enforced by suitably modifying the values:

- ▶ Smallest absolute floating point value in PDF: 0.000015. Numbers with a smaller absolute value will be replaced with 0.
- ▶ (PDF 1.4, but not newer PDF versions) Largest absolute value which can be expressed as floating point number in PDF: 32767.0. Numbers with a larger absolute value will be replaced with the closest integer.

Violations of the following limits will result in an exception:

- ▶ Although PDFlib doesn't have any static limits regarding the PDF output file size, it must enforce certain limits when generating PDF/A-1, PDF/X-4 and PDF/X-5. See the PDFlib Tutorial for details.
- ▶ Largest allowed numerical value in PDF: 2.147.483.647.
- ▶ Maximum length of hypertext strings: 65535.
- ▶ Maximum length of text strings on the page: 32.763 bytes (i.e. 16.381 characters for CID fonts) if *kerining=false* and *wordspacing=0*; otherwise 4095 characters
- ▶ The following options are limited to a maximum of 8191 list entries: *views*, *namelist*, *polylinelist*, *fieldnamelist*, *itemnamelist*, *itemtextlist*, *children*, *group*
- ▶ Maximum number of indirect objects in a PDF/A-1, PDF/X-4 or PDF/X-5 document: 8.388.607

# 2 General Functions

## 2.1 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 closely modelled after the document/page paradigm, generating documents the »natural« way 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 2.1. Figure 2.1 depicts the nesting of scopes. The function descriptions specify the allowed scope for each function. Calling a function outside of the allowed scopes will trigger an exception. You can query the current scope with the *scope* parameter.

*Cookbook* A full code sample can be found in the *Cookbook* topic `general/function_scopes`.

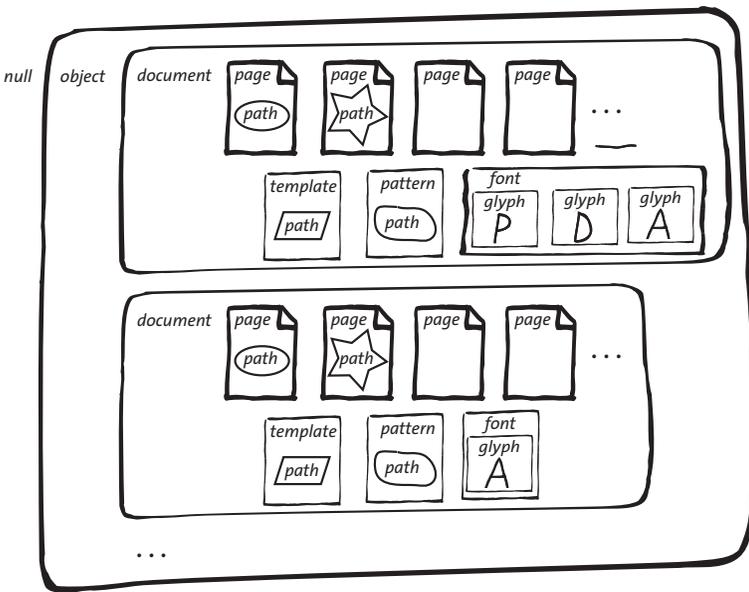


Fig. 2.1  
Nesting of scopes

Table 2.1 Function scope definitions

<b>scope name</b>	<b>definition</b>
<b>path</b>	started by one of <code>PDF_moveto()</code> , <code>PDF_circle()</code> , <code>PDF_arc()</code> , <code>PDF_arcn()</code> , or <code>PDF_rect()</code> , <code>PDF_ellipse()</code> ; terminated by any of the functions in Section 7.5, «Painting and Clipping», page 134
<b>page</b>	between <code>PDF_begin_page_ext()</code> and <code>PDF_end_page_ext()</code> , but outside of path scope
<b>template</b>	between <code>PDF_begin_template_ext()</code> and <code>PDF_end_template_ext()</code> , but outside of path scope
<b>pattern</b>	between <code>PDF_begin_pattern()</code> and <code>PDF_end_pattern()</code> , but outside of path scope
<b>font</b>	between <code>PDF_begin_font()</code> and <code>PDF_end_font()</code> , but outside of glyph scope
<b>glyph</b>	between <code>PDF_begin_glyph()</code> and <code>PDF_end_glyph()</code> , but outside of path scope
<b>document</b>	between <code>PDF_begin_document()</code> and <code>PDF_end_document()</code> , but outside of page, template, pattern, and font scope
<b>object</b>	in object-oriented language bindings: the lifetime of the PDFlib object, but outside of document scope; in other bindings between <code>PDF_new()</code> and <code>PDF_delete()</code> , but outside of document scope
<b>null</b>	outside of object scope
<b>any</b>	when a function description mentions any scope it actually means any except null, since a PDFlib object doesn't even exist in null scope.

## 2.2 Parameter and Option Handling

PDFlib's operation can be controlled by a variety of global parameters. There are string parameters and numerical values for controlling PDFlib and the appearance of the PDF output. Four functions are available for setting and retrieving numerical and string parameters. At the beginning of each section the relevant parameter key names and values are described; 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 get\_value(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** *set\_value(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 get\_parameter(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 language binding: clients must not free the returned string. PDFlib manages all string resources internally.

---

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

**Perl PHP** `set_parameter(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*.

---

**C++ Java** `void set_option(String optlist)`

**Perl PHP** `set_option(string optlist)`

**C** `void PDF_set_option(PDF *p, const char *optlist)`

---

Set one or more global options.

**optlist** An option list specifying global options according to Table 2.2. If an option is provided more than once the last instance will override all previous ones. In order to supply multiple values for a single option (e.g. *searchpath*) supply all values in a list argument to this option. The following options can be used:

*avoiddemo*, *stamp*, *filenamehandling*, *license*, *licensefile*, *logging*, *resourcefile*, *searchpath*, *shutdownstrategy*

**Details** Except for *searchpath*, the new value will override the old one. `PDF_set_option()` supports a subset of the parameters of `PDF_set_parameter()`.

**Scope** *any*

Table 2.2 Options for `PDF_set_option()`

<b>option</b>	<b>description</b>
<b>avoiddemo-stamp</b>	(Boolean) If <code>true</code> , an exception will be thrown when no valid license key was found; if <code>false</code> , a demo stamp will be created on all pages. This option must be set before the first call to <code>PDF_begin_document()</code> . Default: <code>false</code>

Table 2.2 Options for PDF\_set\_option()

option	description
<b>filename-handling</b>	<p>(Keyword; not required on Windows) Target encoding for file names (default: auto on i5/iSeries, otherwise legacy):</p> <p><b>ascii</b> 7-bit ASCII</p> <p><b>basicebcdic</b> Basic EBCDIC according to code page 1047, but only Unicode values &lt;= U+007E</p> <p><b>basicebcdic_37</b> Basic EBCDIC according to code page 0037, but only Unicode values &lt;= U+007E</p> <p><b>honorlang</b> (Not on i5/iSeries) The environment variable LANG is interpreted and applied to file names if it specifies utf8, UTF-8, cpXXXX, CPXXXX, iso8859-x, or ISO-8859-x.</p> <p><b>legacy</b> Use host encoding to interpret the file name and interpret the LANG variable if the honorlang parameter is set.</p> <p><b>unicode</b> Unicode encoding in (EBCDIC-) UTF-8 format</p> <p><b>all valid encoding names</b> Any (internal or user-defined) encoding recognized by PDFlib (see Table 4.3) except glyphid and builtin</p>
<b>license</b>	(String) License key for PDFlib, PDFlib+PDI, or PPS (see PDFlib Tutorial for details). The key can be set before the first call to PDF_begin_document(). Use the avoiddemo stamp option to make sure that missing license keys will not accidentally result in a demo stamp.
<b>licensefile</b>	(Name string) Name of a file containing the license key (see PDFlib Tutorial for details). The license file can only be set once before the first call to PDF_begin_document().
<b>logging</b>	(Option list) Logging options according to Table 2.8
<b>resourcefile</b>	(Name string) 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.
<b>searchpath</b>	(List of name strings) Relative or absolute path name(s) of a directory containing files to be read. The search path can be set multiply; the entries will be accumulated and used in least-recently-set order (see PDFlib Tutorial for details). An empty name string (i.e. {} ) deletes all existing search path entries. On Windows the search path can also be set via a registry entry. Default: platform-specific, see PDFlib Tutorial
<b>shutdown-strategy</b>	<p>(Integer) Strategy for releasing global resources which are allocated once for all PDFlib objects. Each global resource is initialized on demand when it is first needed. This option must be set to the same value for all PDF objects in a process; otherwise the behavior is undefined (default: 0):</p> <p><b>0</b> A reference counter keeps track of how many PDFlib objects use the resource. When the last PDFlib object is deleted and the reference counter drops to zero, the resource is released.</p> <p><b>1</b> The resource is kept until the end of the process. This may slightly improve performance, but requires more memory after the last PDFlib object is deleted.</p>

## 2.3 Setup

Table 2.3 and Table 2.4 list relevant parameter and value key names for PDFlib setup (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 2.3 Setup-related keys for `PDF_get/set_parameter()`

key	explanation
<b>any resource category name</b>	Entries in any of the resource categories. <code>PDF_get_parameter()</code> : 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
<b>asciifile</b>	(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
<b>filename-handling</b>	Target encoding for file names according to Table 2.2
<b>honorlang</b>	(Deprecated, use <code>filenamehandling=honorlang</code> instead) If true, the environment variable LANG will be interpreted and applied to file names if it specifies <code>utf8</code> , <code>UTF-8</code> , <code>cp1252</code> , <code>CP1252</code> , <code>iso8859-x</code> , or <code>ISO-8859-x</code> . Default: false. Scope: object
<b>license<sup>1</sup></b>	(Deprecated) Set the license key for PDFlib, PDFlib+PDI, or PPS. The key can be set before the first call to <code>PDF_begin_document()</code> . Use the <code>nodemostamp</code> parameter to make sure that missing license keys will not accidentally result in a demo stamp. Scope: object
<b>licensefile</b>	(Deprecated) Set the name of a file containing the license key. The license file can only be set once before the first call to <code>PDF_begin_document()</code> . Scope: object
<b>nodemostamp</b>	(Deprecated) 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 <code>PDF_begin_document()</code> . Default: false. Scope: object
<b>resourcefile</b>	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
<b>scope<sup>2</sup></b>	Name of the current scope (see Table 2.1). Scope: any
<b>SearchPath</b>	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 (including the default entries). <code>PDF_get_parameter()</code> : 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 encoded in UTF-8. Default: platform-specific, see PDFlib Tutorial. Scope: any
<b>string<sup>2</sup></b>	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
<b>version<sup>2</sup></b>	Full PDFlib version string in the format <code>&lt;major&gt;.&lt;minor&gt;.&lt;revision&gt;</code> , possibly suffixed with additional qualifiers such as <code>beta</code> , <code>rc</code> , etc. Scope: any, null <sup>3</sup>

1. Only for `PDF_set_parameter()`

2. Only for `PDF_get_parameter()`

3. May be called with a `PDF *` argument of `NULL` or `o`

Table 2.4 Setup-related keys for `PDF_get/set_value()`

key	explanation
<b>compress</b>	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
<b>major minor revision<sup>1</sup></b>	Major, minor, or revision number of PDFlib, respectively. Scope: any, null <sup>2</sup>
<b>maxfile-handles</b>	(Unsupported; implemented on Windows only) New maximum for the number of simultaneously open files (in the C runtime). The number must be greater or equal than 20 and less or equal than 2048. An exception will be thrown if the new value is not accepted by the C runtime. Scope: object

1. Only for `PDF_get_value()`

2. May be called with a `PDF *` argument of `NULL` or 0

---

### 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.

Objective-C: this function is called when the PDFlib object's `init` method is called.

C++, Java, Perl, PHP: 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` sections.

**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.

---

### 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.

Objective-C: this function is called when the PDFlib object's *release* method is called.

Perl and PHP: this function is automatically called when the PDFlib object goes out of scope.

## 2.4 PDFlib Virtual File System (PVF)

*Cookbook* A full code sample can be found in the *Cookbook* topic `general/starter_pvf`.

---

**C++** `void create_pvf(string filename, const void *data, size_t size, string optlist)`

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

**Perl PHP** `create_pvf(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. The name of the virtual file will be subject to the *SearchPath* mechanism if it uses only slash '/' characters as directory or file name separators.

**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 area containing the data.

**optlist** An option list according to Table 2.5. 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 2.5 Options for *PDF\_create\_pvf()*

option	description
<b>copy</b>	(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 delete\_pvf(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; will be interpreted according to the global *filenamehandling* option or parameter, see Table 2.2) 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 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*

## 2.5 Exception Handling

Table 2.6 lists relevant options for this section. These options are supported by many functions as indicated in the corresponding option list descriptions. These options can also be used as parameter key name for `PDF_get/set_parameter()` (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 2.6 Exception-related keys for `PDF_get/set_parameter()` (also available as option)

key	explanation
<b>errorpolicy</b>	(Keyword) Controls the behavior of various functions in case of an error. The parameter <code>errorpolicy</code> can be overridden by the <code>errorpolicy</code> option of many functions, and serves as default for this option. Supported keywords (default: <code>legacy</code> ; scope: any): <b>legacy</b> (Deprecated) The behavior of the functions is the same as in PDFlib 6. <b>return</b> If an error occurs the function will return. Functions which can return an error code (e.g. <code>PDF_load_image()</code> ) return -1 (in PHP: 0). Functions which return result strings (e.g. <code>PDF_fit_table()</code> ) return the string <code>_error</code> . Application developers must check the return value against -1 (in PHP: 0) or <code>_error</code> to detect error situations. In case of an error a detailed description can be queried with <code>PDF_get_errmsg()</code> . This setting is recommended for new applications. <b>exception</b> If an error occurs, the function will throw an exception. The exception must be caught in client code using a binding-specific mechanism. The partial PDF output generated so far will be unusable and must be discarded.

---

**C++ Java** `int get_errnum()`

**Perl PHP** `int get_errnum()`

**C** `int PDF_get_errnum(PDF *p)`

---

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

**Returns** The error code of the most recent error condition.

**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, Objective-C, and PHP this function is also available as `get_errnum()` in the `PDFlibException` object.

---

**C++ Java** `String get_errmsg()`

**Perl PHP** `string get_errmsg()`

**C** `const char *PDF_get_errmsg(PDF *p)`

---

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

**Returns** Text containing the description of the most recent error condition.

**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, Objective-C, and PHP this function is also available as `get_errmsg()` in the `PDFlibException` object.

---

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

**Perl PHP** *string* `get_apiname()`

**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 API 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 Objective-C, and PHP 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.

---

## 2.6 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 2.7 lists the parameter key names for the logging feature (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 2.7 Logging-related keys for `PDF_set_parameter()`

key	explanation
<code>logging</code>	Option list with logging options according to Table 2.8
<code>userlog</code>	String which will be copied to the log file

The logging options can be supplied in the following ways:

- ▶ As an option list for the `logging` option of `PDF_set_option()`, e.g.:  

```
p.set_option("logging", "filename={debug.log} remove")
```
- ▶ 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 2.8 Options for the logging parameter

option	description								
<code>(empty list)</code>	Enable log output								
<code>disable</code>	(Boolean) Disable logging output								
<code>enable</code>	(Boolean) Enable logging output								
<code>filename</code>	(String) Name of the log file; <code>stdout</code> and <code>stderr</code> will be recognized as special names. On CICS this option will be ignored, and logging output will always be written to <code>stderr</code> . Output will be appended to any existing contents. Default: <table border="0"><tr><td><code>pdflog</code></td><td>on z/OS</td></tr><tr><td><code>PDFlib.log</code></td><td>on Mac and iSeries</td></tr><tr><td><code>\PDFlib.log</code></td><td>on Windows</td></tr><tr><td><code>/tmp/PDFlib.log</code></td><td>on all other systems</td></tr></table> The log file name can alternatively be supplied in an environment variable called <code>PDFLIBLOGFILE</code> .	<code>pdflog</code>	on z/OS	<code>PDFlib.log</code>	on Mac and iSeries	<code>\PDFlib.log</code>	on Windows	<code>/tmp/PDFlib.log</code>	on all other systems
<code>pdflog</code>	on z/OS								
<code>PDFlib.log</code>	on Mac and iSeries								
<code>\PDFlib.log</code>	on Windows								
<code>/tmp/PDFlib.log</code>	on all other systems								
<code>flush</code>	(Boolean) If <code>true</code> , 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 <code>false</code> , the log file will be opened only once. Default: <code>false</code>								
<code>remove</code>	(Boolean) If <code>true</code> , an existing log file will be deleted before writing new output. Default: <code>false</code>								
<code>stringlimit</code>	(Integer) Limit for the number of characters per line, or 0 for unlimited. Default: 0								

Table 2.8 Options for the logging parameter

<b>option</b>	<b>description</b>
<b>classes</b>	(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. The following options are provided (default: {api=1 warning=1}):
<b>api</b>	Log all API calls with their function parameters and results. If api=2 a timestamp will be created in front of all API trace lines, and deprecated functions and options will be marked. If api=3 try/catch calls will be logged (useful for debugging problems with nested exception handling).
<b>filesearch</b>	Log all attempts related to locating files via SearchPath or PVF.
<b>resource</b>	Log all attempts at locating resources via Windows registry, UPR definitions as well as the results of the resource search.
<b>user</b>	User-specified logging output supplied with the userlog parameter.
<b>warning</b>	Log all PDFlib warnings, i.e. error conditions which can be ignored or fixed internally. If warning=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 Document and Page Functions

## 3.1 Document Functions

---

C++ Java `int begin_document(String filename, String optlist)`  
Perl PHP `int begin_document(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; will be interpreted according to the global *filenamehandling* option or parameter, see Table 2.2) 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. The supplied *writeproc* must be a C-style function, not a C++ method.

**optlist** An option list specifying document options:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Document options according to Table 3.1. Options specified in *PDF\_end\_document()* have precedence over identical options specified in *PDF\_begin\_document()*. The following options can be used:  
*attachmentpassword, attachments, autoxmp, compatibility, destination, filemode, flush, groups, inmemory, labels, lang, linearize, masterpassword, metadata, moddate, objectstreams, openmode, optimize, pagelayout, pdfa, pdfx, permissions, recordsize, rolemap<sup>2</sup>, 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, but instead of writing to a disk file it calls a client-supplied callback function to deliver the PDF output data. The 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, JavaScript: 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++.

---

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

**Perl PHP** `end_document(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 processing options:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Document options according to Table 3.1. Options specified in `PDF_end_document()` have precedence over identical options specified in `PDF_begin_document()`. The following options can be used:  
*action*, *attachmentpassword*, *attachments*, *autoxmp*, *createpvf*, *destination*, *destname*, *labels*, *metadata*, *moddate*, *objectstreams*, *openmode*, *pagelayout*, *portfolio*, *rolemap*<sup>2</sup>, *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 3.1 Document options for `PDF_begin_document()` and `PDF_end_document()`

option	description
<b>action</b> <sup>1</sup>	(Action list; not allowed for PDF/A) List of document actions for one or more of the following events. Default: empty list. <b>open</b> 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. <b>didprint/didsave/willclose/willprint/willsave</b> (PDF 1.4) JavaScript actions to be performed after printing/after saving/before closing/before printing/ before saving the document.
<b>attachment-password</b> <sup>2,3</sup>	(String; PDF 1.6; will be ignored if <i>userpassword</i> or <i>masterpassword</i> are set; can not be combined with the <i>linearize</i> and <i>optimize</i> options) File attachments will be encrypted using the supplied string as password. The rest of the document will not be encrypted.

Table 3.1 Document options for PDF\_begin\_document() and PDF\_end\_document()

option	description
<b>attachments</b>	(List of option lists; not allowed for PDF/A and PDF/X-1a/3) 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: <b>description</b> (Hypertext string; PDF 1.6) Descriptive text associated with the file. <b>filename</b> (Hypertext string; required) Name of the file. Unicode file names are supported, but require PDF 1.7 for correct display in Acrobat. <b>mimetype</b> (String) MIME type of the file; Acrobat will use it for launching the appropriate application when the attachment is activated. <b>name</b> (Hypertext string) Name of the attachment. Default: filename without any path components
<b>autoxmp</b>	(Boolean; will be forced to true in PDF/X-3/4/5 and PDF/A-1 modes) If true, PDFlib will create XMP document metadata from document info fields (see Section 14.2, »XMP Metadata«, page 227). Default: false
<b>compatibility<sup>2</sup></b>	(Keyword; will be ignored if one of the pdfx or pdfa options is used with a value different from none) Set the document's PDF version to one of the keywords listed below. This option affects which PDF creation features are available and which PDF documents can be imported with PDFlib+PDI (default: 1.7): <b>1.3</b> PDF 1.3 requires Acrobat 4 or above. <b>1.4</b> PDF 1.4 requires Acrobat 5 or above. <b>1.5</b> PDF 1.5 requires Acrobat 6 or above. <b>1.6</b> PDF 1.6 requires Acrobat 7 or above. <b>1.7</b> PDF 1.7 is specified in ISO 32000-1 and requires Acrobat 8 or above. <b>1.7ext3</b> PDF 1.7 extension level 3 requires Acrobat 9 or above. <b>1.7ext8</b> PDF 1.7 extension level 8 requires Acrobat X.
<b>createpvf<sup>2</sup></b>	(Boolean) If true, generate the PDF file in memory instead of on file. The supplied file name is the name of a virtual file which will be created with the call of PDF_end_document(). In this case PDF_get_buffer() cannot be called to fetch the PDF output data; instead, the name of the generated PVF file can be supplied to other PDFlib functions. This may be useful when generating documents which will be included in a PDF Portfolio. Default: false
<b>destination</b>	(Option list; will be ignored if an open action has been specified) An option list specifying the document open action according to Table 12.5.
<b>destname<sup>1</sup></b>	(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.
<b>filemode<sup>2</sup></b>	(String, z/OS and USS only) Parameter string for setting the file mode of the document file and any temporary file (e.g. with the linearize option). The supplied string will be appended to the default file mode of »wb«,«. The option recordsize must be consistent with the parameters specified in this option. Example string: recfm=fb,lrec1=80,space=(cyl,(1,5). Default: empty, or recfm=v for unblocked output.
<b>flush<sup>2</sup></b>	(Keyword; only for PDF_begin_document_callback()) Set the flushing strategy. Default: page. <b>none</b> flush only once at the end of the document <b>page</b> flush at the end of each page <b>content</b> flush after all fonts, images, file attachments, and pages <b>heavy</b> always flush when the internal 64 KB document buffer is full
<b>groups<sup>2</sup></b>	(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 assigned to a page group.

Table 3.1 Document options for `PDF_begin_document()` and `PDF_end_document()`

<b>option</b>	<b>description</b>
<b><code>inmemory</code></b> <sup>2</sup>	(Boolean; not for <code>PDF_begin_document_callback()</code> ) If true and the <code>linearize</code> or <code>optimize</code> option is true as well, PDFlib will not create any temporary files for linearization, but will process the file in memory. This can result in tremendous performance gains on some systems (especially z/OS), but requires memory twice the size of the document. If false, a temporary file will be created for linearization and optimization. Default: false
<b><code>labels</code></b>	(List of option lists) A list containing one or more option lists according to Table 3.2 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 must be unique within a document. Default: none
<b><code>lang</code></b> <sup>2</sup>	(String; recommended if <code>tagged=true</code> ) 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.
<b><code>linearize</code></b> <sup>2</sup>	(Boolean; not for <code>PDF_begin_document_callback()</code> ) If true, the output document will be linearized. On z/OS this option cannot be combined with an empty filename. Default: false
<b><code>master-password</code></b> <sup>2,3</sup>	(String; required if permissions has been specified; 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
<b><code>metadata</code></b>	(Option list; PDF 1.4) Supply XMP document metadata (see Section 14.2, »XMP Metadata«, page 227). The XMP will overwrite document info entries supplied with <code>PDF_set_info()</code> . In PDF/A mode the supplied XMP metadata must conform to additional requirements (see PDFlib Tutorial).
<b><code>moddate</code></b>	(Boolean) If true, the <code>ModDate</code> (modification date) document info key will be created for compliance with some preflight tools. Default: false
<b><code>objectstreams</code></b> <sup>2</sup>	(List of keywords; PDF 1.5; will be forced to false if <code>optimize</code> or <code>linearize</code> is true) Generate compressed object streams which significantly reduce output file size (default: {other nodocinfo}): <b><code>bookmarks</code></b> Compress bookmark objects. <b><code>docinfo</code></b> Compress document info fields. <b><code>fields</code></b> Compress form fields. <b><code>names</code></b> Compress objects for named destinations. <b><code>none</code></b> Don't generate any compressed object streams (except for categories which are explicitly enabled after this option). <b><code>other</code></b> All categories which are not explicitly disabled after this keyword, plus other object types which don't have their own keyword. <b><code>pages</code></b> Compress the objects comprising the page tree. <b><code>tags</code></b> Compress marked content tags. <b><code>xref</code></b> Generate a compressed xref stream. This category will automatically be enabled if at least one of the other categories is enabled. Except for none and other, all keywords can be prefixed with no (e.g. nodocinfo) to disable compression for the specified category. If at least one such negative keyword is supplied, the keyword other will be prepended to the list.
<b><code>openmode</code></b>	(Keyword) Set the appearance when the document is opened. Default: bookmarks if the document contains any bookmarks, otherwise none. <b><code>none</code></b> Open with no additional panel visible. <b><code>bookmarks</code></b> Open with the bookmark panel visible. <b><code>thumbnails</code></b> Open with the thumbnail panel visible. <b><code>fullscreen</code></b> Open in fullscreen mode (does not work in the browser). <b><code>layers</code></b> (PDF 1.5) Open with the layer panel visible. <b><code>attachments</code></b> (PDF 1.6) Open with the attachments panel visible.

Table 3.1 Document options for PDF\_begin\_document() and PDF\_end\_document()

option	description
<b>optimize<sup>2</sup></b>	(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. In general optimization will not have any significant effect except for inefficient client code (e.g. loading the same image or ICC profile multiply instead of reusing the handle). On z/OS this option cannot be combined with in-core generation (i.e. an empty filename). Default: false
<b>pagelayout</b>	(Keyword) The page layout to be used when the document is opened. Default: default. <b>default</b> The default setting of the Acrobat viewer. <b>singlepage</b> Display one page at a time. <b>onecolumn</b> Display the pages continuously in one column. <b>twocolumnleft</b> Display the pages in two columns, odd pages on the left. <b>twocolumnright</b> Display the pages in two columns, odd pages on the right <b>twopageleft</b> (PDF 1.5) Display the pages two at a time, with odd-numbered pages on the left. <b>twopageright</b> (PDF 1.5) Display the pages two at a time, with odd-numbered pages on the right.
<b>pdfa<sup>2</sup></b>	(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. PDF/A-1 output can at the same time conform to the PDF/X-1a:2003, PDF/X-3:2003, and PDF/X-4 settings of the pdfx option. Default: none
<b>pdfx<sup>2</sup></b>	(Keyword) Set the PDF/X conformance level to one of PDF/X-1a:2001, PDF/X-1a:2003, PDF/X-3:2002, PDF/X-3:2003, PDF/X-4, PDF/X-4p, PDF/X-5g, PDF/X-5pg, or none. Default: none
<b>permissions<sup>2</sup></b>	(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): <b>noprint</b> Acrobat will prevent printing the file. <b>nohighresprint</b> (PDF 1.4) Acrobat will prevent high-resolution printing. If noprint isn't set, printing is restricted to the »print as image« feature which prints a low-resolution rendition of the page. <b>nomodify</b> Acrobat will prevent editing or cropping pages and creating or changing form fields. <b>noassemble</b> (PDF 1.4; implies nomodify) Acrobat will prevent inserting, deleting, or rotating pages and creating bookmarks and thumbnails. <b>noannots</b> Acrobat will prevent creating or changing annotations and form fields. <b>noforms</b> (PDF 1.4; implies nomodify and noannots) Acrobat will prevent form field filling. <b>nocopy</b> Acrobat will prevent copying and extracting text or graphics; the accessibility interface will be controlled by noaccessible. <b>noaccessible</b> (PDF 1.4) Acrobat will prevent extracting text or graphics for accessibility purposes (such as a screenreader program). <b>plainmetadata</b> (PDF 1.5) Keep XMP document metadata unencrypted even for encrypted documents.
<b>portfolio<sup>1</sup></b>	(Option list; PDF 1.7) Suboptions for creating a PDF portfolio according to Table 12.16
<b>recordsize<sup>2</sup></b>	(Integer; z/OS and USS 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)
<b>rolemap<sup>2</sup></b>	(List of string lists; the first element in each string list is a name string, the second element is a string; only for Tagged PDF; required if custom element types are used) Mapping of custom element types to standard element types. Each sublist contains the name of a standard or custom element type, and the name of the standard element type to which the first type will be mapped. Inline-level and pseudo element types are not allowed for the second entry in a sublist since custom types cannot be mapped to these types. Standard element type names also can be mapped to other standard element types in order to assign different semantics to existing element types. See Section 14.3, »Tagged PDF«, page 228, regarding the use of custom element types in Tagged PDF.

Table 3.1 Document options for `PDF_begin_document()` and `PDF_end_document()`

option	description
<b>search</b>	(Option list) Instruct Acrobat to attach a search index when opening the document. The following suboptions are supported: <b>filename</b> (Hypertext 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. <b>indextype</b> (Name string) The type of the index; must be PDX for Acrobat. Default: PDX
<b>tagged<sup>2</sup></b>	(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 14.3, »Tagged PDF«, page 228). If the <code>pdfa</code> option has the value »PDF/A-1a:2005« this option will automatically be forced to true. Default: false
<b>tempdirname<sup>2</sup></b>	(String; not for <code>PDF_begin_document_callback()</code> ) Directory where temporary files 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 <code>tempfilenames</code> option has been supplied. Default: empty
<b>tempfilenames<sup>2</sup></b>	(List of two strings; only for z/OS and USS and for <code>PDF_begin_document()</code> ) 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 <code>PDF_end_document()</code> . If this option is supplied the <code>filename</code> parameter must not be empty. Default: empty
<b>uri</b>	(String) Set the document's base URL. This is useful when a document with relative Web links is moved to a different location. Adjusting the base URL makes sure that relative links will still work. Default: none
<b>user-password<sup>2,3</sup></b>	(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
<b>viewer-preferences</b>	(Option list) Option list specifying various viewer preferences according to Table 3.3. Default: empty

1. Only for `PDF_end_document()`

2. Only for `PDF_begin_document()` and `PDF_begin_document_callback()`

3. Characters outside of Winansi encoding are only allowed in passwords if PDF 1.7 extension level 3 or higher is generated.

Table 3.2 Suboptions for the `labels` option in `PDF_begin/end_document()` and `label` option in `PDF_begin/end_page_ext()`

option	description
<b>group</b>	(String; only for <code>PDF_begin_document()</code> ; 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 <code>groups</code> option in <code>PDF_begin_document()</code> .
<b>hypertext-encoding</b>	(Keyword) Specifies the encoding for the <code>prefix</code> option. An empty string is equivalent to unicode. Default: value of the global <code>hypertextencoding</code> parameter.
<b>pagenumber</b>	(Integer; only for <code>PDF_end_document()</code> ; 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.

Table 3.2 Suboptions for the labels option in PDF\_begin/end\_document() and label option in PDF\_begin/end\_page\_ext()

option	description
<b>prefix</b>	(Hypertext string) The label prefix for all labels in the range. Default: none
<b>start</b>	(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
<b>style</b>	(Keyword) The numbering style to be used. Default: none. <b>none</b> no page number; labels will only consist of the prefix. <b>D</b> decimal arabic numerals (1, 2, 3, ...) <b>R</b> uppercase roman numerals (I, II, III, ...) <b>r</b> lowercase roman numerals (i, ii, iii, ...) <b>A</b> uppercase letters (A, B, C, ..., AA, BB, CC, ...) <b>a</b> lowercase letters (a, b, c, ..., aa, bb, cc, ...)

Table 3.3 Suboptions for the viewerpreferences option in PDF\_begin\_document() and PDF\_end\_document()

option	description
<b>centerwindow</b>	(Boolean) Specifies whether to position the document's window in the center of the screen. Default: false
<b>direction</b>	(Keyword) The reading order of the document, which affects the scroll ordering in double-page view and the side (left/right) of the first page for double-page layout in Acrobat (default l2r): <b>l2r</b> Left to right <b>r2l</b> Right to left (including vertical writing systems)
<b>displaydoctitle</b>	(Boolean) Specifies whether to display the Title document info field in Acrobat's title bar (true) or the file name (false). Default: false
<b>duplex</b>	(Keyword; PDF 1.7) Paper handling option for the print dialog (default: none): <b>DuplexFlipShortEdge</b> Duplex and flip on the short edge of the sheet. <b>DuplexFlipLongEdge</b> Duplex and flip on the long edge of the sheet. <b>none</b> No paper handling specified. <b>Simplex</b> Print single-sided.
<b>fitwindow</b>	(Boolean) Specifies whether to resize the document's window to the size of the first page. Default: false

Table 3.3 Suboptions for the viewerpreferences option in PDF\_begin\_document() and PDF\_end\_document()

<b>option</b>	<b>description</b>
<b>hidemenubar</b> <sup>1</sup>	(Boolean) Specifies whether to hide Acrobat's menu bar. Default: false
<b>hidetoolbar</b> <sup>1</sup>	(Boolean) Specifies whether to hide Acrobat's tool bars. Default: false
<b>hidewindow-ui</b> <sup>1</sup>	(Boolean) Specifies whether to hide Acrobat's window controls. Default: false
<b>nonfullscreen-pagemode</b>	(Keyword; only relevant if the openmode option is set to fullscreen) Specifies how to display the document on exiting full-screen mode. Default: none
	<b>bookmarks</b> display page and bookmark pane
	<b>thumbnails</b> display page and thumbnail pane
	<b>layers</b> display page and layer pane
	<b>none</b> display page only
<b>numcopies</b>	(Integer in the range 1-5, PDF 1.7) The number of copies for the print dialog. Default: viewer-specific
<b>picktrayby-pdfsize</b>	(Boolean; PDF 1.7; no effect on Mac OS) Specifies whether the PDF page size is used to select the input paper tray in the print dialog. Default: viewer-specific
<b>printscaling</b>	(Keyword; PDF 1.6) Page scaling option to be selected when a print dialog is presented for the document. Supported keywords (default: appdefault):
	<b>none</b> No page scaling; this may be useful for printing page contents at their exact sizes.
	<b>appdefault</b> Use the current print scaling as specified in Acrobat.
<b>printpage-range</b>	(List with pairs of integers; PDF 1.7) Page numbers for the print dialog. Each pair denotes the start and end page numbers of a page range to be printed (first page is 1). Default: viewer-specific
<b>printarea</b>	(Keyword; for PDF/X only media and bleed are allowed) 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. Supported keywords (default: crop):
<b>printclip</b>	
<b>viewarea</b>	
<b>viewclip</b>	<b>art</b> Use the ArtBox
	<b>bleed</b> Use the BleedBox
	<b>crop</b> Use the CropBox
	<b>media</b> Use the MediaBox
	<b>trim</b> Use the TrimBox

1. Acrobat 8 and above does not support the combination of hidemenubar, hidetoolbar, and hidewindowui (i.e. all user interface elements hidden). The menu bar will still be visible if all three elements are set to hidden.

## 3.2 Fetching PDF Documents from Memory

If a non-empty *filename* parameter has been supplied to *PDF\_begin\_document()* PDFlib will write PDF documents to a named disk file. Alternatively, PDF document data will be generated in memory if the *filename* parameter is empty. In this case the PDF document data must be fetched from memory with *PDF\_get\_buffer()*. This is especially useful when shipping PDF from a Web server.

---

**C++** *const char \*get\_buffer(long \*size)*

**Java** *byte[] get\_buffer()*

**Perl PHP** *string get\_buffer()*

**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. 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.

### 3.3 Page Functions

Table 3.4 and Table 3.5 list relevant parameter and value key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 3.4 Page-related keys for `PDF_get/set_parameter()`

key	explanation
<b>topdown</b>	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 3.5 Page-related keys for `PDF_get_value()`

key	explanation
<b>pagewidth</b> <b>pageheight</b>	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** `begin_page_ext(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 3.6. The PDFlib Tutorial lists applicable page size limits in Acrobat. See also Table 3.7 for more details (options *width* and *height*).

Table 3.6 Common standard page size dimensions in points<sup>1</sup>

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

1. More information about ISO, Japanese, and U.S. standard formats can be found at [www.cl.cam.ac.uk/~mgk25/iso-paper.html](http://www.cl.cam.ac.uk/~mgk25/iso-paper.html)

**optlist** An option list according to Table 3.7. These options have lower priority than identical options specified in `PDF_end_page_ext()`. The following options can be used: *action*, *artbox*, *bleedbox*, *cropbox*, *defaultcmyk1*, *defaultgray1*, *defaultrgb1*, *duration*, *group*, *height*, *label*, *mediabox*, *metadata*, *pagenumber*, *rotate*, *separationinfo*, *taborder*, *topdown*, *transition*, *transparencygroup*, *trimbox*, *userunit*, *viewports*, *width*

**Details** This function resets 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** `end_page_ext(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 3.7. 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, defaultcmyk1, defaultgray1, defaultrgb1, duration, group, height, label, mediabox, metadata, rotate, taborder, transition, transparencygroup, trimbox, userunit, viewports, width*

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

Table 3.7 Page options for `PDF_begin_page_ext()` and `PDF_end_page_ext()`

option	description
<b>action</b>	(Action list) List of page actions for one or more of the following events (default: empty list): <b>open</b> Actions to be performed when the page is opened. <b>close</b> Actions to be performed when the page is closed.
<b>artbox</b> <b>bleedbox</b> <b>cropbox</b>	(Rectangle) Specify the ArtBox, BleedBox, or CropBox for the current page, respectively. The coordinates are specified in the default coordinate system. Default: no box entries
<b>defaultgray</b> <sup>1</sup> <b>defaultrgb</b> <sup>1</sup> <b>defaultcmyk</b> <sup>1</sup>	(ICC handle) Set a default gray, RGB, or CMYK color space for the page according to the supplied profile handle.
<b>duration</b>	(Float) Set the page display duration in seconds for the current page if <code>openmode=fullscreen</code> (see Table 3.1). Default: 1
<b>group</b> <sup>1</sup>	(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 <code>PDF_resume_page()</code> . The group name must have been defined with the <code>groups</code> option in <code>PDF_begin_document()</code> .
<b>height</b>	(Float or keyword; not allowed if the <code>topdown</code> option or parameter is true) The dimensions of the new page in points (or user units, if the <code>userunit</code> option has been specified). In order to produce landscape pages use <code>width &gt; height</code> or the <code>rotate</code> option. PDFlib uses <code>width</code> and <code>height</code> to construct the page's MediaBox, but the MediaBox can also explicitly be set using the <code>mediabox</code> option. The <code>width</code> and <code>height</code> options will override the parameters with the same name. The following symbolic page size names can be used as keywords by appending <code>.width</code> or <code>.height</code> (e.g. <code>a4.width</code> , <code>a4.height</code> ): <code>a0, a1, a2, a3, a4, a5, a6, b5, letter, legal, ledger, 11x17</code>
<b>label</b>	(Option list) An option list according to Table 3.2 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 <code>style/prefix/start</code> values must be unique within a document.

Table 3.7 Page options for `PDF_begin_page_ext()` and `PDF_end_page_ext()`

<b>option</b>	<b>description</b>
<b>mediabox</b>	(Rectangle; not allowed if the <code>topdown</code> option or parameter is <code>true</code> ) Change the <code>MediaBox</code> for the current page. The coordinates are specified in the default coordinate system. By default, the <code>MediaBox</code> will be created by using the width and height parameters. The <code>mediabox</code> option will override the width and height options and parameters.
<b>metadata</b>	(Option list; PDF 1.4) Supply metadata for the page (see Section 14.2, »XMP Metadata«, page 227)
<b>pagenumber<sup>1</sup></b>	(Integer) If this option is specified with a value <code>n</code> , the page will be inserted before the existing page <code>n</code> within the page group specified in the <code>group</code> 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.
<b>rotate</b>	(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
<b>separation-info<sup>1</sup></b>	(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 pre-separated workflow: <ul style="list-style-type: none"> <li><b>pages</b> (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.</li> <li><b>spotname</b> (String; required unless <code>spotcolor</code> has been supplied) The name of the colorant for the current page.</li> <li><b>spotcolor</b> (Spot color handle) A color handle describing the colorant for the current page.</li> </ul>
<b>taborder</b>	(Keyword; PDF 1.5) Keyword specifying the tab order for form fields and annotations on the page (Default: <code>none</code> ): <ul style="list-style-type: none"> <li><b>column</b> Form fields and annotations are visited column by column from top to bottom, where columns are ordered as specified by the <code>direction</code> suboption of the <code>viewerpreferences</code> option of <code>PDF_begin/end_document()</code>.</li> <li><b>none</b> The tab order is unspecified.</li> <li><b>structure</b> Form fields and annotations are visited in the order in which they appear in the structure tree. The order for annotations that are not included in the structure tree is unspecified.</li> <li><b>row</b> Form fields and annotations are visited row by row starting at the topmost row, where the direction within a row is as specified by the <code>direction</code> suboption of the <code>viewerpreferences</code> option of <code>PDF_begin/end_document()</code>.</li> </ul>
<b>topdown<sup>1</sup></b>	(Boolean) If <code>true</code> , the origin of the coordinate system at the beginning of the page will be assumed in the top left corner of the page, and <code>y</code> coordinates will increase downwards; otherwise the default coordinate system will be used. Default: <code>false</code>

Table 3.7 Page options for PDF\_begin\_page\_ext() and PDF\_end\_page\_ext()

option	description
<b>transition</b>	(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 3.1). Default: replace <b>split</b> Two lines sweeping across the screen reveal the page <b>blinds</b> Multiple lines sweeping across the screen reveal the page <b>box</b> A box reveals the page <b>wipe</b> A single line sweeping across the screen reveals the page <b>dissolve</b> The old page dissolves to reveal the page <b>glitter</b> The dissolve effect moves from one screen edge to another <b>replace</b> The old page is simply replaced by the new page <b>fly</b> (PDF 1.5) The new page flies into the old page. <b>push</b> (PDF 1.5) The new page pushes the old page off the screen <b>cover</b> (PDF 1.5) The new page slides on to the screen and covers the old page. <b>uncover</b> (PDF 1.5) The old page slides off the screen and uncovers the new page. <b>fade</b> (PDF 1.5) The new page gradually becomes visible through the old one.
<b>transparency-group</b>	(Option list; PDF 1.4; not allowed for PDF/A, PDF/X-1, and PDF/X-3; certain rules apply for PDF/X-4 and PDF/X-5, see PDFlib Tutorial) Specifies transparency group attributes for the generated page, an imported page, or a template. Supported options: <b>colorspace</b> (Keyword; required) Specifies the color space of the transparency group with one of the following keywords: DeviceGray, DeviceRGB, DeviceCMYK. <b>isolated</b> (Boolean) Specifies whether the transparency group is isolated. Default: false <b>knockout</b> (Boolean) Specifies whether the transparency group is a knockout group. Default: false Default: if a page contains image masks with more than 1 bit or the opacityfill/opacitystroke options of PDF_create_gstate() the following option list will automatically be created to improve output quality: transparencygroup={colorspace=DeviceRGB}
<b>trimbox</b>	(Rectangle) Specify the TrimBox for the current page. The coordinates are specified in the default coordinate system. Default: no TrimBox entry
<b>userunit</b>	(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, or m which scales to the respective unit. User units don't change the actual page contents; they are only a hint to Acrobat which is used when printing the page or using the measurement tools. Default: 1 (i.e. one unit is one point)
<b>viewports</b>	(List of option lists; PDF 1.7ext3) Specifies one or more georeferenced areas (viewports) on the page; see Section 13.2, »Geospatial Features«, page 222, for details. Viewports allow different geospatial references (specified by the georeference option) to be used on different areas of the page, e.g. for multiple maps. The ordering of the option lists in the viewports list is relevant for overlapping viewports: the last viewport which contains a point will be used for that point.
<b>width</b>	(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** `suspend_page(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 graphics (graphics, color, text, etc.) and layer state of the current page 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** `resume_page(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 3.8. 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 3.8 Options for `PDF_resume_page()`

<b>option</b>	<b>description</b>
<b>group</b>	(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 <code>groups</code> option in <code>PDF_begin_document()</code> .
<b>pagenumber</b>	(Integer) If this option is supplied, the page with the specified number within the page group chosen in the <code>group</code> 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.

## 3.4 Layers

*Cookbook* A full code sample can be found in the *Cookbook* topic `graphics/starter_layer`.

---

```
C++ Java int define_layer(String name, String optlist)
Perl PHP int define_layer(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 3.9. The following options can be used:

- ▶ General options: and *hypertextencoding* and *hypertextformat* (see Table 12.1)
- ▶ Layer control options:  
*creatorinfo*, *defaultstate*, *initialexportstate*, *initialprintstate*, *initialviewstate*, *intent*,  
*language*, *onpanel*, *pageelement*, *printssubtype*, *removeunused*, *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. Layers which are used on multiple pages should be defined only once (e.g. before creating the first page). If *PDF\_define\_layer()* is called repeatedly on multiple pages, the layer definitions will accumulate (even if they have the same name), which is usually not desired.

PDF/X: Layers are not allowed in PDF/X-1/2/3. Some options are restricted in PDF/X-4/5.

**Scope** *document*, *page*

Table 3.9 Options for *PDF\_define\_layer()*

option	explanation
<b>creatorinfo</b>	(Option list; not for PDF/X) An option list describing the content and the creating application. Both of the following entries are required if this option is used: <b>creator</b> (Hypertext string) The name of the application which created the layer <b>subtype</b> (String) The type of content. Suggested values are <i>Artwork</i> and <i>Technical</i> .
<b>defaultstate</b>	(Boolean) Specifies whether or not the layer will be visible by default. Default: true
<b>initial-exportstate</b>	(Boolean; not for PDF/X) 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
<b>initial-printstate</b>	(Boolean; not for PDF/X) The layer's recommended printing state. If true, Acrobat will include the layer when printing the document. Default: true
<b>initial-viewstate</b>	(Boolean; not for PDF/X) The layer's recommended viewing state. If true, Acrobat will display the layer when opening the document. Default: true
<b>intent</b>	(Keyword) Intended use of the graphics: <i>View</i> or <i>Design</i> . Default: <i>View</i>

Table 3.9 Options for PDF\_define\_layer()

option	explanation
<b>language</b>	(Option list; not for PDF/X) Specifies the language of the layer: <b>lang</b> (String; required) The language and possibly locale in the format described in Table 3.1 for the lang option <b>preferred</b> (Boolean) If true this layer is used if there is only a partial match between the layer and the system language. Default: false
<b>onpanel</b>	(Boolean; not for PDF/X) If false, the layer name will not be visible in Acrobat's layer panel, and therefore cannot be manipulated by the user. Default: true
<b>pageelement</b>	(Keyword; not for PDF/X) 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).
<b>printsubtype</b>	(Option list; not for PDF/X) Specifies whether the layer is intended for printing: <b>subtype</b> (Keyword) One of Trapping, PrintersMarks, or Watermark specifying the kind of content in the layer. <b>printstate</b> (Boolean) If true, Acrobat will activate the layer contents upon printing.
<b>removeunused</b>	(Boolean) If true and the layer is not used on a page, the layer will not be included in the page's layer list. A layer is considered in use on a page if it has been supplied to PDF_begin_layer() at least once on that page. Default: false unless the layer is included in a non-default variant with listmode=visiblepages.
<b>zoom</b>	(List of floats or percentages; not for PDF/X) 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 set_layer_dependency(string type, string optlist)
C void PDF_set_layer_dependency(PDF *p, const char *type, const char *optlist)

```

---

Define layer relationships and variants (requires PDF 1.5).

**type** The type of dependency or relationship according to Table 3.10.

Table 3.10 Dependency and relationship types for layers

type	notes; options specific for this type
<b>GroupAllOn</b>	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
<b>GroupAnyOn</b>	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
<b>GroupAllOff</b>	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
<b>GroupAnyOff</b>	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
<b>Lock</b>	(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
<b>Parent</b>	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

Table 3.10 Dependency and relationship types for layers

type	notes; options specific for this type
<b>Radiobtn</b>	Specify a radiobutton relationship between the layers specified in the <code>group</code> 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. Option specific for this type: <code>group</code>
<b>Title</b>	The layer handle specified in the <code>parent</code> option does not control any page contents directly, but serves as the parent layer node for the layers specified in the <code>children</code> option. Options specific for this type: <code>children</code> , <code>parent</code>
<b>Variant</b>	Specify a document variant, i.e. a combination of one or more layers. Later calls to <code>PDF_set_layer_dependency()</code> can supply the <code>variantname</code> option again in order to specify dependency rules for this configuration. Options specific for this type: <code>basestate</code> , <code>defaultvariant</code> , <code>includelayers</code> , <code>invisiblelayers</code> , <code>visiblayers</code>

**optlist** An option list specifying layer dependencies according to Table 3.11. The following options can be used:

- ▶ General option: *hypertextencoding* (see Table 12.1)
- ▶ Layer dependency options:  
*basestate*, *children*, *createorderlist*, *defaultvariant*, *depend*, *includelayers*, *invisiblelayers*, *group*, *visiblayers*, *listmode*, *parent*, *variantname*.

**Details** Layer relationships specify the presentation of layer names in Acrobat’s layer pane, as well as the visibility of one or more layers when the user interactively enables or disables layers.

Variants can be regarded as a fixed combination of layers to enhance production safety. Instead of manipulating individual layers the user can only enable or disable a variant. If a document contains variants, Acrobat 9 will not display individual layer names but only the names of the variants. Layer variants are presented in Acrobat 9 only for PDF/X documents. Acrobat X does not display layer variants.

In order to specify a dependency in the presence of layer variants where not all affected layers are part of the same variant, the dependency must be specified before setting the default variant.

PDF/X: Layers are not allowed in PDF/X-1/2/3. Some options are restricted in PDF/X-4/5.

**Scope** *document*, *page*; Layer relationships should be specified after all layers have been defined.

Table 3.11 Options for `PDF_set_layer_dependency()`

option	explanation
<b>basestate</b>	(Keyword; only for <code>type=Variant</code> ; not for PDF/X) Specify the visibility of all layers which are not explicitly configured in the <code>visiblayers</code> and <code>invisiblelayers</code> options. Supported keywords (default: <code>on</code> ): <b>on</b> All layers will be visible for the selected variant. <b>off</b> All layers will be invisible for the selected variant. <b>unchanged</b> The state of all layers will be left unmodified for the selected variant.
<b>children</b>	(List of layer handles; only for <code>type=Parent</code> and <code>Title</code> ) One or more layer handles specifying the layers subordinate to the provided parent layer.

Table 3.11 Options for `PDF_set_layer_dependency()`

option	explanation
<b>createorderlist</b>	(Boolean; only for <code>type=Variant</code> and <code>defaultvariant=true</code> ) Include all layers in the <code>/Order</code> array which can be used to present all layers in a user interface. Setting this option to true has the following implications: <ul style="list-style-type: none"> <li>▶ Acrobat 9 displays the layer variants in its Layers panel, but not the layer names. Acrobat 9 emits PDF/X-4 validation errors for documents with <code>createorderlist</code> since this option is not allowed in PDF/X-4:2008.</li> <li>▶ Acrobat X displays the layer names in its Layers panel, but not the layer variants. Acrobat X successfully validates documents with <code>createorderlist</code> since this option is allowed in PDF/X-4:2010.</li> </ul> Default: false for PDF/X without any layer variants, true otherwise
<b>defaultvariant</b>	(Boolean; only for <code>type=Variant</code> ) If true, the specified variant is the default variant, i.e. it will be active when the document is opened. Exactly one variant must be specified as default variant. Default: false
<b>depend</b>	(Layer handle; only for <code>type=GroupAllOn</code> , <code>GroupAnyOn</code> , <code>GroupAllOff</code> , and <code>GroupAnyOff</code> ) The layer which is controlled by the layers specified in the group option.
<b>group</b>	(List of layer handles; only for <code>type=GroupAllOn</code> , <code>GroupAnyOn</code> , <code>GroupAllOff</code> , <code>GroupAnyOff</code> , <code>Lock</code> , and <code>Radiobtn</code> ) One or more layer handles comprising the group. For <code>type=Lock</code> all layers in the group will be locked.
<b>includelayers</b>	(List of layer handles; only for <code>type=Variant</code> ) Specify the layers which belong to the variant. Default: all layers defined so far in the document
<b>invisiblelayers</b>	(List of layer handles; only for <code>type=Variant</code> ) Specify a list of layers which will initially be invisible for the selected variant. A layer must not be listed in a variant's <code>visiblelayers</code> and <code>invisiblelayers</code> lists at the same time. If <code>defaultvariant=true</code> this option overrides the <code>defaultstate</code> option of <code>PDF_define_layer()</code> . Default (depends on the <code>basestate</code> option): all layers in the <code>includelayers</code> list if <code>basestate=off</code> ; empty list if <code>basestate=on</code> ;
<b>listmode</b>	(Keyword; only for <code>type=Variant</code> ) Specify which layer names will be displayed in Acrobat's layer pane. Supported keywords (default: <code>visiblepages</code> ): <ul style="list-style-type: none"> <li><b>allpages</b> The names of all layers on all pages will be displayed.</li> <li><b>visiblepages</b> The names of all layers on the currently visible page(s) will be displayed. This implies the default value <code>removeunused=true</code> for all layers which belong to the variant.</li> </ul> In Acrobat this will have an effect only if <code>defaultvariant=true</code> .
<b>parent</b>	(Layer handle; only for <code>type=Parent</code> and <code>Title</code> ) The layer which is the parent of the layers specified in the <code>children</code> option.
<b>variantname</b>	(Hypertext string; required for <code>type=Variant</code> ) Name of the selected variant. If <code>type=Variant</code> each variant name must be specified only once. Default if type is different from <code>Variant</code> : the default variant
<b>visiblelayers</b>	(List of layer handles; only for <code>type=Variant</code> ) Specify a list of layers which will initially be visible in the selected variant. A layer must not be listed in a variant's <code>visiblelayers</code> and <code>invisiblelayers</code> lists at the same time. If <code>defaultvariant=true</code> this option overrides the <code>defaultstate</code> option of <code>PDF_define_layer()</code> . Default (depends on the <code>basestate</code> option): all layers in the <code>includelayers</code> list if <code>basestate=on</code> ; empty list if <code>basestate=off</code> ;

---

**C++ Java** `void begin_layer(int layer)`

**Perl PHP** `begin_layer(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 be controlled with the *layer* option of the respective functions.

*Scope* *page*

---

**C++ Java** *void end\_layer()*

**Perl PHP** *end\_layer()*

**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*



# 4 Font and Text Functions

## 4.1 Font Handling

Table 4.1 lists relevant parameter and value key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 4.1 Font-related keys for `PDF_get/set_parameter()`

### key and 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 2.3. Scope: any

---

```
C++ Java int load_font(String fontname, String encoding, String optlist)
Perl PHP int load_font(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) Name of the font. It can alternatively be provided via the `fontname` option which will override this parameter. See option `fontname` in Table 4.3 for details.

**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** Name of the encoding. It can alternatively be provided via the `encoding` option which will override this parameter. See option `encoding` in Table 4.3 for details. Note the following list of common encoding-related problems:

- ▶ An 8-bit encoding was supplied but the font does not contain any glyph for this encoding, or the font is a standard CJK font.
- ▶ The encoding `builtin` was supplied, but the font does not contain any internal encoding. This can only happen for TrueType fonts.
- ▶ A predefined CMap was supplied but doesn't match the font.

**optlist** An option list with the following options:

- ▶ General option: `errorpolicy` (see Section 2.5, »Exception Handling«, page 27)
- ▶ Font loading options according to Table 4.3:  
`ascender`, `autocidfont`, `autosubsetting`, `capheight`, `descender`, `dropcorewidths`, `embedding`, `encoding`, `fallbackfonts`, `fontname`, `fontstyle`, `initialsubset`, `keepfont`, `keepnative`, `linegap`, `metadata`, `monospace`, `optimizeinvisible`, `readfeatures`, `readkerning`, `readshaping`, `replacementchar`, `skipposttable`, `subsetlimit`, `subsetminsize`, `subsetting`, `unicodemap`, `vertical`, `xheight`

A font handle for later use with `PDF_info_font()`, text output functions, and the `font` text appearance option. If the requested font/encoding combination cannot be loaded 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 of -1 (in PHP: o) will be returned or an exception raised. The error behavior can be changed with the *errorpolicy* parameter or option.

If the function returns -1 (in PHP: o) you can request the reason of the failure with *PDF\_get\_errmsg()*. Otherwise, the value returned by this function can be used as font handle when calling other font-related functions. The returned handle doesn't have any significance to the user other than serving as a font handle.

The returned font handle is valid until the font is closed with *PDF\_close\_font()*. Finishing the document with *PDF\_end\_document()* closes each open font handle unless the option *keepfont* has been supplied in the respective *PDF\_load\_font()* call, or the font has been loaded in *object* scope (i.e. outside of any document).

**Details** This function prepares a font for later use.

Repeated calls: when this function is called again with the same font name, the same encoding, and the same options, the same font handle as in the first call will be returned. Exceptions: if one of the following options has been specified in the first call, but not in the subsequent call, the second font handle will nevertheless be identical to the first font handle: *embedding*, *readkerning*, *replacementchar*, *fallbackfonts*, *metadata*. Similarly, the *initialsubset* option will be ignored when comparing fonts, e.g. if the font has first been loaded without *initialsubset* and is loaded again with *initialsubset*, a handle to the first font will be returned and *initialsubset* will not have any effect.

Trying to load a font again will fail if *embedding=false* in the first call and *embedding=true* in the second call. This situation usually points to a problem in the application.

Implicit font loading: in addition to explicitly loading a font with *PDF\_load\_font()*, some API functions (e.g. *PDF\_add/create\_textflow()* or *PDF\_fill\_textblock()*) can implicitly load a font for which the font name and encoding have been specified in an option list. A new font handle will be created unless the font has already been loaded earlier.

Some text output features are not available for certain encodings (see Table 4.2).

Table 4.2 Availability of PDFlib features for various encodings

feature	unicode and Unicode CMaps	8-bit encodings	legacy CMaps, cp936 etc.	glyphid
Textflow	yes	yes	yes <sup>1</sup>	yes
glyph replacement	yes <sup>2</sup>	yes	yes <sup>1</sup>	–
fallback fonts	yes <sup>2</sup>	yes	yes <sup>1</sup>	–
shaping	yes <sup>2</sup>	–	yes <sup>1</sup>	yes
OpenType layout features	yes	–	yes <sup>1</sup>	yes

1. This feature is not available for CJK fonts with *keepnative=true*.

2. This feature is not available for standard CJK fonts with *Unicode CMaps* or *keepnative=true*.

**Scope** any

**Params** See Table 4.1.

Table 4.3 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>ascender</b>	(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 <code>PDF_info_font()</code> )
<b>autocidfont</b>	(Boolean) If true, TrueType fonts with 8-bit encodings which are not a subset of winansi regarding the set of Unicode values 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
<b>auto-subsetting</b>	(Boolean) Dynamically decide whether or not the font will be subset, subject to the <code>subsetlimit</code> and <code>subsetminsize</code> options and the actual usage of glyphs. This option will be ignored if the subsetting option has been supplied. Default: true
<b>capheight</b>	(Integer between -2048 and 2048) See ascender above.
<b>descender</b>	(Integer between -2048 and 2048) See ascender above.
<b>dropcore-widths</b>	(Boolean; unsupported; will be forced to false in PDF/A and PDF/X mode) The widths for unembedded core fonts will not be emitted in the generated PDF. This slightly reduces output file size, but may create incorrect text rendering for certain characters. It is strongly recommended to keep this option at its default value. Default: false
<b>embedding</b>	<p>(Boolean; must be true for PDF/A and PDF/X; will be ignored for SING and Type 3 fonts which are always embedded) 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.</p> <p>Default: generally false, but true in certain situations involving TrueType and OpenType fonts with encodings which result in conversion to a CID font. Although PDFlib will automatically embed such fonts, font embedding can be prevented by setting <code>embedding</code> to false. In this case the font must be installed on the system where the PDF documents are viewed or printed.</p> <p>The option <code>embedding=false</code> will be ignored if the same font has already been loaded earlier with <code>embedding=true</code>. The embedding behavior for fonts with invisible text can be modified with the <code>optimizeinvisible</code> option even for <code>embedding=true</code>.</p>

Table 4.3 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>encoding</b>	<p>(String; required for implicit font loading except for <code>PDF_fill_textblock()</code> and if the text appearance option font is not specified) The encoding in which incoming text for this font will be expected (case is significant):</p> <p>Wide character encodings:</p> <ul style="list-style-type: none"> <li>▶ unicode and the names of Unicode CMaps</li> <li>▶ the name of a non-Unicode (legacy) CMap, or Identity-H or Identity-V for CID addressing</li> <li>▶ glyphid: all glyphs in the font can be addressed by their font-specific ID</li> </ul> <p>Byte- and multibyte encodings:</p> <ul style="list-style-type: none"> <li>▶ one of the predefined 8-bit encodings winansi, macroman, macroman_apple, ebcdic, ebcdic_37, pdfdoc, iso8859-X, or cpXXXX, and non-Unicode CMaps</li> <li>▶ 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)</li> <li>▶ host or auto for an automatically selected encoding;</li> <li>▶ the name of a user-defined encoding loaded from file or defined via <code>PDF_encoding_set_char()</code>;</li> <li>▶ builtin to select the font's internal encoding (mostly for symbolic fonts);</li> <li>▶ an encoding name known to the operating system (not available on all platforms)</li> </ul> <p>In non-Unicode language bindings, the <code>textformat=auto</code> behaves as follows (note that all UTF formats are allowed for both cases):</p> <ul style="list-style-type: none"> <li>▶ Wide character encodings: text in the loaded font will be treated with text format utf16 (for <code>encoding=glyphid</code> surrogates will not be interpreted)</li> <li>▶ Byte- and multibyte encodings: text in the loaded font will be treated with text format bytes.</li> </ul> <p><code>PDF_load_font()</code>: this option can alternatively be provided as function parameter.</p> <p><code>PDF_fill_textblock()</code>: this option is required unless the string in the text parameter is empty and the defaulttext property is used, or the font option has been supplied.</p>
<b>fallbackfonts</b>	<p>(List of option lists according to Table 4.4) Specify one or more fallback fonts for the loaded font. Each fallback font must be defined by a font handle in the font suboption or suitable suboptions for implicit font loading. Fallback fonts are not supported for some combinations of font type and encoding (see Table 4.2).</p> <p>If <code>glyphcheck=replace</code> and the text contains a character which is not part of the base font's 8-bit encoding, or the base font does not contain a glyph for the character, or glyph replacement is forced via the <code>forcechars</code> suboption, PDFlib will search a glyph for this character in all specified fallback fonts in the order in which they are listed. If a suitable glyph is found in one of the fallback fonts, the character will be rendered with this glyph; otherwise the usual glyph replacement mechanism applies.</p>
<b>fontname</b>	<p>(Name string; required for implicit font loading except for <code>PDF_fill_textblock()</code> if the text appearance option font is not specified) Real or alias name of the font (case is significant). This name will be used to find the font data. On Windows, font style names can be appended to the font name after a comma (see PDFlib Tutorial for details). If fontname starts with an '@' character the font will be applied in vertical writing mode.</p> <p><code>PDF_load_font()</code>: can alternatively be provided as function parameter.</p>
<b>fontstyle</b>	<p>(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 <code>fakebold</code> parameter or option). In the latter case the slanting angle can be controlled with the <code>italicangle</code> parameter or option. If this option is applied to one of the core fonts, the appropriate bold, italic, or bold-italic 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</p>

Table 4.3 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>initialsubset</b>	<p>(List of Unichars or Unicode ranges, or list of keywords; only relevant for <code>embedding=true</code> and <code>subsetting=true</code>) Specify the Unicode values for which glyphs will be included in the initial font subset. This can be used to reduce the font size in PDF while still creating identical subsets, which in turn facilitates later optimizations when merging multiple documents. The Unicode values can be specified explicitly by Unichars or Unicode ranges, or implicitly by the name of an 8-bit encoding. Unichars and Unicode ranges have precedence over encoding names. Supported keywords (default: empty) :</p> <p><b>empty</b> The initial font subset will be empty; the contents of the subset will be determined by the text in the document.</p> <p><b>any 8-bit encoding name</b> All Unicode values found in the encoding will be included in the initial subset. Glyphs for additional characters will be added to the subset automatically if required by the text in the document or by the features and shaping text options.</p>
<b>keepfont</b>	<p>(Boolean; not allowed for Type 3 fonts) If false the font will be deleted automatically in <code>PDF_end_document()</code>. If true the font can also be used in subsequent documents until <code>PDF_close_font()</code> has been called. Default: true if <code>PDF_load_font()</code> is called in object scope, otherwise false</p>
<b>keepnative</b>	<p>(Boolean; only relevant for unembedded CJK fonts with a predefined CMap; will be ignored for other fonts; will be forced to false if <code>embedding=true</code>) If false, text in this font will be converted to CID values 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). However, the font can be used in Textflow and all simple text output functions (but not in form fields). Except for glyph replacement and fallback fonts which are unavailable, a font with Unicode CMaps will behave as with <code>encoding=unicode</code>.</p> <p>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: false for TrueType fonts or <code>embedding=true</code>, and true otherwise.</p>
<b>kerning</b>	<p>Deprecated, use the <code>readkerning</code> option to control parsing of kerning data from the font.</p>
<b>linegap</b>	<p>(Integer between -2048 and 2048) See <code>ascender</code> above.</p>
<b>metadata</b>	<p>(Option list; PDF 1.4) Supply metadata for the font (see Section 14.2, »XMP Metadata«, page 227)</p>
<b>monospace</b>	<p>(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)</p>
<b>optimize-invisible</b>	<p>(Boolean; not for PDF/X-1/2/3) If true, fonts which are exclusively used for invisible text (i.e. <code>text-rendering=3</code>) will not be embedded even if <code>embedding=true</code>. This may be useful to avoid font embedding for PDF/A output with invisible text containing OCR results. Even if the font is not embedded, font files must be configured as usual since PDFlib decides about non-embedding only at the end of the document. Default: false</p>
<b>readfeatures</b>	<p>(Boolean; only relevant for TrueType and OpenType fonts and <code>encoding=unicode</code>, <code>glyphid</code>, or Unicode CMaps) Specifies whether the feature tables of a TrueType or OpenType font will be read from the font. Actually applying OpenType features to text is controlled by the <code>features text</code> option (see Table 5.3). Setting this option to false may speed up font loading if OpenType features are not required. Default: true</p>
<b>readkerning</b>	<p>(Boolean) Controls whether or not kerning values will be read from the font. Actually applying the kerning values to text is controlled by the <code>kerning text</code> option (see Table 5.3). Setting this option to false may speed up font loading if kerning is not required. Default: true</p>
<b>readshaping</b>	<p>(Boolean; only relevant for TrueType and OpenType fonts and the encodings <code>unicode</code> and <code>glyphid</code>) Specifies whether the shaping tables of a TrueType or OpenType font will be read, which is a requirement for complex script shaping. Actually shaping text is controlled by the <code>shaping text</code> option (see Table 5.3). Setting this option to false can save memory if shaping is not required. Default: true</p>

Table 4.3 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>replacementchar</b>	<p>(Unichar; only relevant if <code>glyphcheck=replace</code>; ignored for fonts loaded with a non-Unicode CMap or <code>glyphid</code> encoding) Glyphs which are not available in the selected font and which cannot be substituted by fallback fonts or typographically similar characters will be replaced with the specified Unicode value. <code>U+0000</code> can be used to specify the font's »missing glyph« symbol; however, this is not allowed for the PDF/A-1, PDF/X-4, and PDF/X-5 standards. For symbolic fonts loaded with <code>encoding=builtin</code> the code must be supplied instead of the Unicode value.</p> <p>Default: <code>U+00A0</code> (NO-BREAK SPACE) if available in the font; otherwise <code>U+0020</code> (SPACE) if available in the font, otherwise <code>U+0000</code> (but not for PDF/A-1, PDF/X-4, and PDF/X-5). These values will also be used if the font doesn't contain any glyph for the specified <code>replacementchar</code>.</p>
<b>skippost-table</b>	<p>(Boolean; unsupported; only relevant for TrueType and OpenType fonts) Specifies whether the post table of TrueType/OpenType fonts will be parsed to determine glyph names. Setting this option to <code>true</code> can speed up font loading, but glyph name references to glyphs with non-standard names will not work for the font (this mainly affects symbolic fonts, but usually not text fonts). Default: <code>false</code></p>
<b>subsetlimit</b>	<p>(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: <code>100%</code></p>
<b>subsetminsize</b>	<p>(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: <code>50</code></p>
<b>subsetting</b>	<p>(Boolean) Controls whether or not the font will be subset. Subsetting for Type 3 fonts requires a two-pass definition of the font (see PDFlib Tutorial), and the subsetting option must be provided in the first call to <code>PDF_load_font()</code>. Default: <code>false</code></p>
<b>unicodemap</b>	<p>(Boolean; must not be set to <code>false</code> for <code>pdfa=PDF/A-1a:2005</code>) Controls the generation of ToUnicode CMaps. This option will be ignored in Tagged PDF mode. Default: <code>true</code></p>
<b>vertical</b>	<p>(Boolean; only for TrueType and OpenType fonts; will be ignored for predefined CMaps, and will be forced to <code>true</code> if the font name starts with <code>@</code>) If <code>true</code>, the font will be prepared for vertical writing mode.</p>
<b>xheight</b>	<p>(Integer between <code>-2048</code> and <code>2048</code>) See <code>ascender</code> above.</p>

Table 4.4 Suboptions for the fallbackfonts option of PDF\_load\_font()

option	description
<b>font loading options</b>	If the font is specified implicitly (i.e. via the fontname and encoding options, as opposed to the font option), all font loading options according to Table 4.3 except fallbackfonts can be supplied as suboptions. Fonts loaded with a non-Unicode CMap can not be used as fallback fonts.
<b>font</b>	(Font handle) A font handle returned by a call to PDF_load_font() without the fallbackfonts option. If this option is supplied, all font loading options (including fontname and encoding) will be ignored. The font must not be a standard CJK font with embedding=false and keepnative=true.
<b>fontsize</b>	(Float or percentage) Size of the fallback font in user coordinates or as a percentage of the current font size. This option can be used to adjust the size of the fallback font if the design size of the fallback font doesn't match that of the base font. Default: 100%
<b>forcechars</b>	(List of Unichars or Unicode ranges, or single keyword) Specify characters which will always be rendered with glyphs from the fallback font (regardless of the glyphcheck setting). The fallback font must contain glyphs for the specified characters (if individual characters are specified), or at least glyphs for the first and last characters in the specified Unicode range. Unicode values can be specified for this option even if an 8-bit encoding has been specified for the base font. The following keyword can be supplied: <b>gaiji</b> The fallback font must refer to a SING font, and this keyword can be used as a shortcut for the Unicode value of the main glyph of the SING font.
<b>textrise</b>	(Float or percentage) Text rise value (see Table 5.2). Percentages are based on the size specified for the fallback font (i.e. after applying the fontsize suboption if present). This option can be used to adjust the position of text in the fallback font if the design size of the fallback font doesn't match that of the base font. Default: 0

---

**C++ Java** void close\_font(int font)  
**Perl PHP** close\_font(int font)  
**C** void PDF\_close\_font(PDF \*p, int font)

---

Close an open font handle which has not yet been used in the document.

**font** A font handle returned by PDF\_load\_font() which has not already been used in the document or closed.

**Details** This function closes a font handle, and releases all resources related to the font. The font handle must not be used after this call. Usually fonts will automatically be closed at the end of a document. However, closing a font is useful in the following situations:

- ▶ After querying font properties with PDF\_info\_font() it was determined that the font will not be used in the current PDF document.
- ▶ A font was retained across document boundaries (with the keepfont option of PDF\_load\_font()), but now it should be disposed because it is no longer required.

If the font has already been used in the current document it must not be closed.

**Scope** any

---

```

C++ Java double info_font(int font, String keyword, String optlist)
Perl PHP float info_font(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()`, or -1 (in PHP: 0) for some keywords.

**keyword** A keyword specifying the requested information according to Table 4.6. The following keywords can be used:

- ▶ Keywords for glyph mapping: *cid*, *code*, *glyphid*, *glyphname*, *unicode*
- ▶ Font metrics: *ascender*, *capheight*, *descender*, *italicangle*, *linegap*, *xheight*
- ▶ Font file, name, and type: *cidfont*, *familyname*, *fontfile*, *fontname*, *fonttype*, *metricsfile*, *outlineformat*, *singfont*, *standardfont*, *supplement*
- ▶ Technical font information: *feature*, *featurelist*, *fontstyle*, *hostfont*, *kerningpairs*, *monospace*, *numglyphs*, *shapingsupport*, *vertical*
- ▶ Font/encoding relationship: *codepage*, *codepagelist*, *encoding*, *fallbackfont*, *keepnative*, *maxcode*, *numcids*, *numusableglyphs*, *predefcmap*, *replacementchar*, *symbolfont*, *unicodfont*, *unmappedglyphs*
- ▶ Results of font processing for the current document: *numusedglyphs*, *usedglyph*, *willembed*, *willsubset*

**optlist** An option list which additionally qualifies the selected keyword. The following options can be used:

- ▶ Keyword-specific options which are detailed along with the corresponding keyword in Table 4.6.
- ▶ Mapping options according to Table 4.5 for specifying glyphs: *cid*, *code*, *glyphid*, *glyphname*, *unicode*. These options define the source value for the mapping keywords *cid*, *code*, *glyphid*, *glyphname*, and *unicode*. The mapping options are mutually exclusive. The *code*, *glyphname*, and *unicode* options can be combined with the *encoding* option.

Table 4.5 Options for specifying glyphs in `PDF_info_font()`

option	description
<i>cid</i>	(Number) CID value of the glyph; only reasonable if <i>cidfont</i> =1
<i>code</i>	(Number in the range 0...255; only for fonts with 8-bit encoding) Encoding slot
<i>glyphid</i>	(Number in the range 0...65535) Internal glyph id
<i>glyphname</i>	(String) Name of a glyph; not reasonable if <i>cidfont</i> =1
<i>unicode</i>	(Unichar) Unicode character

**Returns** The value of some font or encoding 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 2.3).

**Details** This function supplies information from the following distinct sources:

- ▶ If a valid font handle is supplied it returns information gathered from the font. Examples: font metrics, name, or type; *unicode* value for a particular *glyphid*.
- ▶ If *font* = -1 (in PHP: 0) and the *encoding* option is supplied it returns information about this encoding. Example: *unicode* value for a *code* in the encoding.
- ▶ If *font* = -1 (in PHP: 0) and the *encoding* option is not supplied it returns information gathered from PDFlib's internal tables. Example: *unicode* value for a particular *glyphname*.

**Scope** any except *object*

Table 4.6 Keywords and options for `PDF_info_font()`

<b>keyword</b>	<b>explanation and options</b>
<b>ascender</b>	Metrics value for the ascender. Supported options (default: <code>fontsize=1000</code> ): <b>faked</b> (Boolean) 1 if the value had to be estimated because it was not available in the font or metrics file, otherwise 0 <b>fontsize</b> (FontSize) Value will be scaled to the specified font size
<b>capheight</b>	Metrics value for the capheight. See ascender.
<b>cid</b>	CID for the specified glyph, or -1 if not available. Supported options: <code>cid</code> , <code>glyphid</code> , <code>unicode</code>
<b>cidfont</b>	1 if the font will be embedded as a CID font, otherwise 0
<b>code</b>	Number in the range 0...255 specifying an encoding slot or -1 if no such slot was found in the font or in the encoding (if the <i>encoding</i> option was supplied and <i>font</i> =-1 (in PHP: 0)). Supported options are the mapping options <code>code</code> , <code>glyphid</code> , <code>glyphname</code> , <code>unicode</code> plus the following: <b>encoding</b> (String) Name of an 8-bit encoding
<b>codepage</b>	Check whether the font supports a specific codepage. The information will be taken from the OS/2 table of TrueType/OpenType fonts if available. Supported option: <b>name</b> (String; required) Specifies the name of a codepage in the form <code>cpXXXX</code> , where <code>XXXX</code> indicates the decimal number of a codepage (e.g. <code>cp437</code> , <code>cp1252</code> ) The following return values indicate whether the specified codepage is supported by the font: <b>-1</b> Unknown because the font is not a TrueType or OpenType font. <b>0</b> Font does not support the specified codepage. <b>1</b> Font supports the specified codepage.
<b>codepagelist</b>	String index of a space-separated list of all codepages supported by the font (in the form <code>cpXXXX</code> ), or -1 if the codepage list is unknown because the font is not a TrueType or OpenType font (see <i>codepage</i> ).
<b>descender</b>	Metrics value for the descender. See ascender.
<b>encoding</b>	String index of the name of the font's encoding or CMap. Supported options (default: <code>actual</code> ): <b>api</b> (Boolean) If <code>true</code> , the encoding name as specified in the API <b>actual</b> (Boolean) If <code>true</code> , the name of the actual encoding used for the font
<b>fallbackfont</b>	Handle of the base or fallback font which will be used to render the character specified in the <i>unicode</i> option. This can be used to check which font in the chain of fallback fonts actually provides the glyph used for the specified character. If the character cannot be rendered with any of the base or fallback fonts, -1 will be returned. Supported option: <i>unicode</i>
<b>familyname</b>	String index of the name of the font family, or -1 if unavailable

Table 4.6 Keywords and options for `PDF_info_font()`

<b>keyword</b>	<b>explanation and options</b>
<b>feature</b>	<p>Check whether the font contains a specific OpenType feature table which is supported by PDFlib. Supported options:</p> <p><b>language</b> (String; only if script is supplied) Specifies the language name.</p> <p><b>name</b> (String; required) Specifies the four-character name of an OpenType feature table, e.g. liga (standard ligatures), ital (italic forms in CJK fonts), vert (vertical writing). Feature kern can not be queried.</p> <p><b>script</b> (String) Specifies the script name.</p> <p>The following return values indicate whether the specified OpenType feature table is present in the font and supported by PDFlib:</p> <p>-1 No feature tables available in the font.</p> <p>0 The feature table is not available or not supported by PDFlib.</p> <p>1 The feature table is available for the specified script and language and is supported by PDFlib.</p>
<b>featurelist</b>	String index of a space-separated list of all features which are available in the font and supported by PDFlib, or -1 if no feature tables are available.
<b>fontfile</b>	String index of the path name for the font outline file, or -1 if unavailable
<b>fontname</b>	String index of the font name, or -1 if unavailable. Supported options (default: acrobat):
<b>api</b>	(Boolean) If true, the font name as specified in the API
<b>full</b>	(Boolean) If true, the /FontName entry in the PDF font descriptor
<b>acrobat</b>	(Boolean) If true, the font name as displayed in Acrobat
<b>fontstyle</b>	String index for the value of the fontstyle option (normal, bold, italic, or bolditalic). Supported option:
<b>faked</b>	1 if fontstyle will be realized by PDFlib, 0 if fontstyle will be realized by Acrobat
<b>fonttype</b>	String index of the font type, or -1 if unavailable. Known font types are Multiple Master, OpenType, TrueType, TrueType (CID), Type 1, Type 1 (CID), Type 1 CFF, Type 1 CFF (CID), Type 3
<b>glyphid</b>	Number in the range 0...65535 specifying the font-internal id (GID) of the specified glyph, or -1 if no such glyph was found. Supported options are the mapping options cid, code, glyphid, glyphname, unicode.
<b>glyphname</b>	String index of the name of the specified glyph, or -1 if no such glyph was found in the font or in the specified encoding (if the encoding option was supplied and font=-1 (in PHP: o)). Supported options are the mapping options code, glyphid, glyphname, unicode plus the following:
<b>encoding</b>	(String) Name of an 8-bit encoding
<b>hostfont</b>	1 if the font is a host font, 0 otherwise
<b>italicangle</b>	Italic angle of the font (ItalicAngle in the PDF font descriptor)
<b>keepnative</b>	The resulting value of the keepnative option.
<b>kerningpairs</b>	Number of kerning pairs in the font
<b>linegap</b>	Metrics value for the linegap. See ascender.
<b>maingid</b>	Glyph ID of the main glyph (member mainGID of SING table).
<b>maxcode</b>	Highest code value for the font's encoding, in particular: 0xFF for single-byte encodings, numglyphs-1 for encoding=glyphid, and the highest Unicode value in the encoding otherwise.
<b>metricsfile</b>	String index of the path name for the font metrics file (AFM or PFM), or -1 if unavailable
<b>monospace</b>	The value of the monospace option, or 0 if it hasn't been supplied.
<b>numcids</b>	Number of CIDs if the font uses a standard CMap, otherwise -1

Table 4.6 Keywords and options for `PDF_info_font()`

<b>keyword</b>	<b>explanation and options</b>
<b>numglyphs</b>	Number of glyphs in the font (including the <code>.notdef</code> glyph). Since GIDs start at 0 the highest possible GID value is one smaller than <code>numglyphs</code> .
<b>numusable-glyphs</b>	Number of glyphs in the font which can be reached by the encoding supplied in <code>PDF_load_font()</code>
<b>numused-glyphs</b>	Number of glyphs used in generated text so far.
<b>outlineformat</b>	Font format; one of PFA, PFB, LWFN, TTF, OTF, TTC
<b>predefcmap</b>	String index of the name of a predefined CMap which was specified as encoding for the font, or -1 if unavailable.
<b>replacementchar</b>	Unicode value of the character specified in the <code>replacementchar</code> option. For symbolic fonts loaded with <code>encoding=builtin</code> the code will be returned instead of the Unicode value.
<b>shaping-support</b>	1 if the font supports shaping and the <code>readshaping</code> option was supplied in <code>PDF_load_font()</code> , otherwise 0
<b>singfont</b>	1 if the font is a SING (gaiji) font, otherwise 0
<b>standard-font</b>	1 if the font is a PDF core font or a standard CJK font, otherwise 0
<b>supplement</b>	Supplement number of the character collection for fonts with a standard CJK CMap, otherwise 0
<b>symbolfont</b>	1 if the font is a symbolic font, 0 otherwise (symbol flag in the PDF font descriptor)
<b>unicode</b>	Unicode UTF-32 value for the specified glyph, or -1 if no Unicode value was found in the font or encoding (if the encoding option was supplied and <code>font=-1</code> (in PHP: 0)). Supported options are the mapping options <code>cid</code> , <code>code</code> , <code>glyphid</code> , <code>glyphname</code> , <code>unicode</code> plus the following: <b>encoding</b> (String) Name of an 8-bit encoding
<b>unicodefnt</b>	1 if the font/encoding combination provides Unicode mapping for the glyphs, otherwise 0. CJK fonts with non-Unicode CMaps and <code>keepnative=true</code> will return 0.
<b>unmapped-glyphs</b>	Number of glyphs in the font which are mapped to Unicode PUA values, regardless of whether the PUA value was already present in the font or has been assigned by PDFlib.
<b>usedglyph</b>	1 if the specified glyph ID was used in the text, otherwise 0. Supported option: <code>glyphid</code>
<b>vertical</b>	1 if the font is for vertical writing mode, otherwise 0
<b>weight</b>	Font weight in the range 100...900; 400=normal, 700=bold
<b>willembded</b>	1 if the font will be embedded (via the embedding option or forced font embedding), otherwise 0
<b>willsubset</b>	1 if a font subset will be created (if <code>autosubsetting=true</code> , the <code>subsetlimit</code> must be reached for subsetting to be activated), otherwise 0
<b>xheight</b>	Metrics value for the <code>xheight</code> . See <code>ascender</code> .

## 4.2 Type 3 Font Definition

*Cookbook* A full code sample can be found in the *Cookbook* topic `fonts/starter_type3font`.

```
C++ Java void begin_font(String fontname,  
double a, double b, double c, double d, double e, double f, String optlist)  
Perl PHP begin_font(string fontname, float a, float b, float c, float d, float e, float f, string optlist)  
C void PDF_begin_font(PDF *p, const 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 of the font definition for Type 3 font subsets) 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 4.7. 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. The font can be used until the end of the current *document* scope.

**Scope** *document*; 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 4.7 Options for `PDF_begin_font()`

option	description
<b>colorized</b>	(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
<b>familyname<sup>1</sup></b>	(String; PDF 1.5) Name of the font family
<b>stretch<sup>1</sup></b>	(Keyword; PDF 1.5) The font stretch value. Keywords: ultracondensed, extracondensed, condensed, semicondensed, normal, semiexpanded, expanded, extraexpanded, ultraexpanded. Default: normal
<b>weight<sup>1</sup></b>	(Integer or keyword; PDF 1.5) The font weight. Possible numbers or equivalent keywords are 100=thin, 200=extralight, 300=light, 400=normal, 500=medium, 600=semibold, 700=bold, 800=extrabold, 900=black. Default: normal

Table 4.7 Options for `PDF_begin_font()`

option	description
<b>widthonly</b>	(Boolean) If <code>true</code> (pass 1 for Type 3 font subsetting), only the metrics of the font and glyphs will be defined. No other API functions should be called between <code>PDF_begin_glyph()</code> and <code>PDF_end_glyph()</code> . If other functions are called nevertheless, they will not have any effect on the PDF output, and will not raise any exception. If <code>widthonly=false</code> (pass 2 for Type 3 font subsetting) the actual glyph outlines can be defined. This two-pass definition enables PDFlib to perform subsetting on Type 3 fonts. Default: <code>false</code>

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

<b>C++ Java</b>	<code>void end_font()</code>
<b>Perl PHP</b>	<code>end_font()</code>
<b>C</b>	<code>void PDF_end_font(PDF *p)</code>

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.

<b>C++ Java</b>	<code>void begin_glyph(String glyphname, double wx, double llx, double lly, double urx, double ury)</code>
<b>Perl PHP</b>	<code>begin_glyph(string glyphname, float wx, float llx, float lly, float urx, float ury)</code>
<b>C</b>	<code>void PDF_begin_glyph(PDF *p, const char *glyphname, double wx, double llx, double lly, double urx, double ury)</code>

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. It is strongly recommended to use glyph names according to the Adobe Glyph List (AGL). Glyph names within a font must be unique.

**wx** (Will be ignored in the second pass of the font definition for Type 3 font subsets) 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 of the font definition for Type 3 font subsets) 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*).

PDFlib will determine the Unicode value for each glyph by searching the glyph name in its internal list. If the glyph name isn't found, a PUA Unicode value will be assigned to the glyph name (starting with U+E000). This value can be queried with `PDF_info_font()`.

*Scope* *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** `end_glyph()`

**C** `void PDF_end_glyph(PDF *p)`

---

Terminate a glyph definition for a Type 3 font.

*Scope* *glyph*; this function changes from *glyph* scope to *font* scope, and must always be paired with a matching *PDF\_begin\_glyph()* call.

## 4.3 Encoding Definition

---

C++ Java `void encoding_set_char(String encoding, int slot, String glyphname, int uv)`  
Perl PHP `encoding_set_char(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  $0 \leq \text{slot} \leq 255$ . 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.

It is strongly recommended to supply each glyph name/Unicode value only once in an encoding (with the exception of `.notdef/U+0000`). If slot 0 is used, it should contain the `.notdef` character.

If the encoding is intended for use with Type 3 fonts it is recommended to specify the encoding slots only with glyph names.

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

*Scope* any

## 4.4 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 4.8 and Table 4.9 lists relevant parameters and values for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 4.8 Text-related keys for `PDF_get/set_parameter()`

key	explanation
<b>autospace</b>	If true and the current font contains a glyph for U+0020, PDFlib will automatically add a space character after each text output generated with a <code>show</code> operation. This may be useful for generating Tagged PDF. Note that adding spaces changes the current text position after the <code>show</code> operation. Default: false. Scope: any
<b>charref</b>	See Table 5.1. Unlike the option with the same name this parameter also affects hypertext strings and name strings. Default: false. Scope: any
<b>decoration-above</b>	See Table 5.2. Default: false. Scope: any
<b>escape-sequence</b>	See Table 5.1. Unlike the option with the same name this parameter also affects hypertext strings and name strings. For example, Windows UNC file names must start with four backslash characters if <code>escapesequence=true</code> . Warning: this parameter also affects environment variables (e.g. <code>PDFLIB-LICENSEFILE</code> ). Since path names in environment variables on Windows usually contain backslash characters this parameter should better be avoided on Windows system; supply the option of the same name to specific functions instead. Default: false. Scope: any
<b>fakebold</b>	See Table 5.2. Default: false. Scope: page, pattern, template, glyph, document
<b>glyphcheck</b>	See Table 5.1. Default: replace. Scope: any
<b>kerning</b>	See Table 5.2. Default: true. Scope: any
<b>textformat</b>	(Only for non Unicode-aware language bindings) The format in which the text output functions will expect the client-supplied strings. Supported keywords are <code>bytes</code> , <code>utf8</code> , <code>ebcdicutf8</code> (only on <code>iSeries</code> and <code>zSeries</code> ), <code>utf16</code> , <code>utf16le</code> , <code>utf16be</code> , and <code>auto</code> . Default: <code>auto</code> . Scope: any
<b>underline overline strikeout</b>	See Table 5.2. Default: false. Scope: page, pattern, template, glyph, document

Table 4.9 Text-related keys for `PDF_get/set_value()`

key	explanation
<b>charspacing</b>	See Table 5.2. Only float values are supported. Scope: page, pattern, template, glyph, document
<b>font<sup>1</sup></b>	Identifier of the current font which has been set with <code>PDF_setfont()</code> , or -1 (in PHP: 0) if no font is set. Scope: page, pattern, template, glyph
<b>fontsize<sup>1</sup></b>	Size of the current font which must have been previously set with <code>PDF_setfont()</code> . Scope: page, pattern, template, glyph
<b>horizscaling</b>	See Table 5.2. Only float values are supported. Scope: page, pattern, template, glyph, document
<b>italicangle</b>	See Table 5.2. Scope: page, pattern, template, glyph, document

Table 4.9 Text-related keys for `PDF_get/set_value()`

key	explanation
<b>leading</b>	Leading, which is the distance between baselines of adjacent lines of text. The leading is used for <code>PDF_continue_text()</code> . It is set to the value of the font size when a new font is selected using <code>PDF_setfont()</code> . 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
<b>strokewidth</b>	See Table 5.2. Only float values are supported. The value 0 designates a built-in default. Default: 0
<b>textrendering</b>	See Table 5.2. Scope: page, pattern, template, glyph, document
<b>textrise</b>	See Table 5.2. Only float values are supported. Scope: page, pattern, template, glyph
<b>textx<sup>1</sup></b> <b>texty<sup>1</sup></b>	The x or y coordinate of the current text position. Default: 0. Scope: page, pattern, template, glyph
<b>underline-position</b>	See Table 5.2. Only float values specifying a fraction of the font size are supported. 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
<b>underline-width</b>	See Table 5.2. Only float values are supported. The value 0 uses a font-specific value from the font metrics or outline file if available, otherwise 5% of the fontsize. Default: 0
<b>wordspacing</b>	See Table 5.2. Only float values are supported. Scope: page, pattern, template, glyph, document

1. Only for `PDF_get_value()`

---

**C++ Java** `void PDF_setfont(int font, double fontsize)`  
**Perl PHP** `setfont(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** This function sets the font which will be used by low-level text output functions, e.g. `PDF_show()`. 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.

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

---

Set the position for simple 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 4.8 and Table 4.9.

---

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

**Perl PHP** ***show(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 4.8 and Table 4.9.

*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 4.8 and Table 4.9.

*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** `show_xy(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 at the specified position.

**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 4.8 and Table 4.9.

**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** `continue_text(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_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 4.8 and Table 4.9.

**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 stringwidth(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). If character spacing has been specified, it will be applied after the last glyph as well (this behavior differs from *PDF\_info\_textline()*).

**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 4.8 and Table 4.9.

**Bindings** *PDF\_stringwidth2()* is only available in C since in all other bindings arbitrary string contents can be supplied with *PDF\_stringwidth()*.

## 4.5 Unicode Conversion Functions

These functions may be useful for Unicode string conversion. They are provided for the benefit of users working in environments that do not provide such converters.

**Bindings** The Unicode conversion functions are not available in Unicode-aware language bindings (except C++).

C binding: the strings returned by the functions in this chapter will be stored in a ring buffer with up to 10 entries. If more than 10 strings are converted, the buffers will be re-used, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this function can be used as parameters for a *printf()* statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

---

**C++** *string utf16\_to\_utf8(string utf16string)*

**Perl PHP** *string utf16\_to\_utf8(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 ( $\backslash\text{EF}\backslash\text{BB}\backslash\text{BF}$ ) unless it contains only characters in the range  $\lt U+007F$ . On EBCDIC platforms the conversion result including the BOM will finally be converted to EBCDIC.

**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 utf8\_to\_utf16(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 ( $\backslash\text{FF}\backslash\text{FE}$ ).

- ▶ *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.

**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 utf32\_to\_utf16(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. 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.

**Scope** *any*

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

---

**C++** *string utf8\_to\_utf32(string utf8string, string ordering)*

**Perl PHP** *string utf8\_to\_utf32(string utf8string, string ordering)*

**C** *const char \*PDF\_utf8\_to\_utf32(PDF \*p, const char \*utf8string, const char \*ordering, int \*size)*

---

Convert a string from UTF-8 format to UTF-32.

**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** Reserved, must be empty.

**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-32 string in the platform's native byte order.

**Scope** *any*

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

---

**C++** *string utf32\_to\_utf8(string utf32string)*

**Perl PHP** *string utf32\_to\_utf8(string utf32string)*

**C** *const char \*PDF\_utf32\_to\_utf8(PDF \*p, const char \*utf32string, int len, int \*size)*

---

Convert a string from UTF-32 format to UTF-8.

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

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

**Returns** The converted UTF-8 string. The generated UTF-8 string will start with the UTF-8 BOM (`\xEF\xBB\xBF`) unless it contains only characters in the range `< U+007F`. On EBCDIC platforms the conversion result including the BOM will finally be converted to EBCDIC.

**Scope** *any*

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

---

**C++** *string utf16\_to\_utf32(string utf16string, string ordering)*

**Perl PHP** *string utf16\_to\_utf32(string utf16string, string ordering)*

**C** *const char \*PDF\_utf16\_to\_utf32(PDF \*p, const char \*utf16string, int len, const char \*ordering, int \*size)*

---

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

**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.

**ordering** Reserved, must be empty.

**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-32 string in the platform's native byte order.

**Scope** *any*

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



# 5 Text and Table Formatting

## 5.1 Text Options

This section lists text options for `PDF_fit_textline()`, `PDF_info_textline()`, and `PDF_add/create_textflow()`. Text options also apply to table cells and text Blocks:

- ▶ input filter options according to Table 5.1;
- ▶ text appearance options according to Table 5.2;
- ▶ shaping and typographic options according to Table 5.3;

Table 5.1 Input filter options

option	explanation
<b>charref</b>	(Boolean) If <code>true</code> , enable substitution of numeric and character entity references and glyph name references. Default: the global <code>charref</code> parameter
<b>escape-sequence</b>	(Boolean) If <code>true</code> , enable substitution of escape sequences in content strings, hypertext strings, and name strings. In text for <code>PDF_create_textflow()</code> with <code>inline</code> option lists the begin character for an inline option list can be expressed by an escape sequence. However, escape sequences don't work in inline option lists including the end character. Default: the global <code>escapesequence</code> parameter
<b>glyphcheck</b>	(Keyword) Glyph checking policy: what happens if text contains a code which cannot be mapped to a glyph in the selected font (default: value of the <code>glyphcheck</code> parameter): <b>none</b> No checking <b>error</b> An exception will be thrown for unavailable glyphs. A detailed error message can be retrieved with <code>PDF_get_errmsg()</code> . <b>replace</b> PDFlib will try to replace unavailable glyphs with appropriate replacement glyphs in the base and fallback fonts if available; ligatures will be decomposed. If a suitable replacement is not available, the glyph will be replaced with <code>replacementchar</code> if specified
<b>textformat</b>	(Keyword; only for non Unicode compatible language bindings) Format used to interpret the supplied text. Supported keywords: <code>bytes</code> , <code>utf8</code> , <code>ebcdicutf8</code> (only on <code>iSeries</code> and <code>zSeries</code> ), <code>utf16</code> , <code>utf16le</code> , <code>utf16be</code> , and <code>auto</code> . Default: the global <code>textformat</code> parameter (see Table 4.8)

Table 5.2 Text appearance options

option	explanation
<b>charspacing</b>	(Float or percentage) Character spacing, i.e. the shift of the current point after placing individual characters in a string. Float values specify units of the user coordinate system; percentages are based on <code>fontsize</code> . In order to spread characters apart use positive values for horizontal writing mode, and negative values for vertical writing mode. Default: <code>0</code>
<b>dasharray</b>	(List of two floats) The lengths of dashes and gaps for stroked (outline) text and decoration. Default: <code>{0 0}</code> (i.e. a solid line)
<b>decoration-above</b>	(Boolean) If <code>true</code> , the text decoration enabled with the <code>underline</code> , <code>strikeout</code> , and <code>overline</code> options will be drawn above the text, otherwise below the text. Changing the drawing order affects visibility of the decoration lines. Default: <code>false</code>
<b>fakebold</b>	(Boolean) If <code>true</code> , 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: <code>false</code>

Table 5.2 Text appearance options

option	explanation		
<b>fillcolor</b>	(Color) Fill color of the text. Default for PDF_fit_textline() if inittextstate=false: the corresponding parameter in the current graphics state. Default: {gray 0} (in PDF/A mode: {lab 0 0 0})		
<b>font</b>	(Font handle) Handle for the font to be used. If this option is supplied, all font loading options (including fontname and encoding) will be ignored. Using the font option instead of implicit font loading with the fontname/encoding options offers performance benefits. Default: the implicitly loaded font if available, else the font in the current text state (only for Textline and inittextstate=false). Otherwise no font is available, which will trigger an error.		
<b>fontsize</b>	(FontSize; required) 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). With Textflows percentages relate to the size of the preceding text. Default: the current font size		
<b>gstate</b>	(Gstate handle) Handle for a graphics state retrieved with PDF_create_gstate(). The graphics state affects all text created with this function. Default: no graphics state (i.e. current settings will be used).		
<b>horizscaling</b>	(Float or percentage; must be different from 0) Horizontal text scaling to the given percentage (must be different from 0). Text scaling shrinks or expands the text by a given percentage. Text scaling always relates to the horizontal coordinate. Default <sup>1</sup> : 100%		
<b>italicangle</b>	(Float) 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 <sup>1</sup> : 0		
<b>kerning</b>	(Boolean) If true, enable kerning for fonts which have been opened with the readkerning option; disable otherwise. Default: the global kerning parameter		
<b>overline</b>	(Boolean) Overline mode. Default <sup>1</sup> : false		
<b>strikeout</b>	(Boolean) Strikeout mode. Default <sup>1</sup> : false		
<b>strokecolor</b>	(Color; only effective if textrendering is set to stroke text) Stroke color of the text. Default: see fillcolor		
<b>strokewidth</b>	(Float, percentage, or keyword; only effective if textrendering is set to outline text) Line width for outline text (in user coordinates or as a percentage of the fontsize). The keyword auto or the value 0 uses a built-in default. Default: auto		
<b>textrendering</b>	(Integer) Text rendering mode. Only the value 3 has an effect on Type 3 fonts (default <sup>1</sup> : 0):		
0	 fill text	4	 fill text and add it to the clipping path
1	 stroke text (outline)	5	 stroke text and add it to the clipping path
2	 fill and stroke text	6	 fill and stroke text and add it to the clipping path
3	invisible text	7	 add text to the clipping path (does not have any effect with PDF_fit_textline() or PDF_fit_textflow())
<b>textrise</b>	(Float or percentage) Textrise parameter, which specifies the distance between the desired text position and the baseline. Positive values of textrise move the text up. The textrise always relates to the vertical coordinate. This may be useful for superscripts and subscripts. Percentages are based on fontsize. Default <sup>1</sup> : 0		
<b>underline</b>	(Boolean) Underline mode. Default <sup>1</sup> : false		

Table 5.2 Text appearance options

option	explanation
<b>underline-position</b>	(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 <sup>1</sup> : auto
<b>underline-width</b>	(Float, percentage, or keyword) Line width for underlined text (absolute value or percentage of the fontsize). The keyword auto or the value 0 uses a font-specific value from the font metrics or outline file if available, otherwise 5%. Default <sup>1</sup> : auto
<b>wordspacing</b>	(Float or percentage) Wordspacing, 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 value is specified in user coordinates or a percentage of the fontsize. Default <sup>1</sup> : 0

<sup>1</sup>. Default for PDF\_fit\_textline() and PDF\_info\_textline() if inittextstate=false: the parameter with the same name as the option (see Table 4.8 and Table 4.9)

Table 5.3 Shaping and typographic options

option	explanation
<b>features</b>	(List of keywords) Specifies which typographic features of an OpenType font will be applied to the text, subject to the script and language options. Keywords for features which are not present in the font will silently be ignored. The following keywords can be supplied: <b>_none</b> Apply none of the features in the font. As an exception, the vert feature must explicitly be disabled with the novert keyword. <b>&lt;name&gt;</b> Enable a feature by supplying its four-character OpenType tag name. Some common feature names are liga, ital, tnum, smcp, swsh, zero. The full list with the names and descriptions of all supported features can be found in the PDFlib Tutorial. <b>no&lt;name&gt;</b> The prefix no in front of a feature name (e.g. noliga) disables this feature. Default: _none for horizontal writing mode, vert for vertical writing mode.
<b>language</b>	(Keyword; only relevant if script is supplied) The text will be processed according to the specified language, which is relevant for the features and shaping options. A full list of keywords can be found in the PDFlib Tutorial, e.g. ARA (Arabic), JAN (Japanese), HIN (Hindi). Default: _none (undefined language)
<b>script</b>	(Keyword; required if shaping=true) The text will be processed according to the specified script, which is relevant for the features, shaping, and advancedlinebreak options. The most common keywords for scripts are the following: _none (undefined script), latn, grek, cyrl, armn, hebr, arab, deva, beng, guru, gujr, orya, tamL, thai, laoo, tibT, hang, kana, han. A full list of keywords can be found in the PDFlib Tutorial. The keyword _auto selects the script to which the majority of characters in the text belong, where latn and _none are ignored. _auto is only relevant for shaping and will be ignored for features and advancedlinebreak. Default: _none
<b>shaping</b>	(Boolean) If true, complex script shaping and bidirectional reordering will be applied to the text according to the script and language options. The script option must have a value different from _none and the font must obey certain conditions (see PDFlib Tutorial). Shaping is only done for characters in the same font. Shaping is not available for right-to-left text in Textflows (only in Textlines). Default: false

Table 5.4 Suboptions for the leader option for PDF\_fit\_textline() and PDF\_add/create\_textflow() and inline options in PDF\_create\_textflow()

<b>option</b>	<b>explanation</b>
<b>font loading options</b>	If the font is specified implicitly (i.e. via the fontname and encoding options, as opposed to the font option), all font loading options according to Table 4.3 can be supplied as suboptions.
<b>alignment</b>	(One or two keywords) Textline: 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 for Textline: {none grid}; default for Textflow: grid): <ul style="list-style-type: none"> <li><b>center</b> Textline: the leader is justified between the Textline and the border of the fitbox. Textflow: 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).</li> <li><b>grid</b> 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.</li> <li><b>justify</b> Textline: the leader is justified between the Textline and the border of the fitbox by applying a suitable character spacing. Textflow: 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 a suitable character spacing.</li> <li><b>left</b> The leader is repeated starting from the left border of the fitbox or the end of the Textline, respectively. This may result in a gap at the start of the Textline or the right border of the fitbox, respectively.</li> <li><b>none</b> No leader</li> <li><b>right</b> The leader is repeated starting from the right border of the fitbox or the beginning of the Textline, respectively. This may result in a gap at the end of the Textline or the left border of the fitbox, respectively.</li> </ul>
<b>fillcolor</b>	(Color) Color of the leader. Default: color of the text line
<b>font</b>	(Font handle) Handle for the font to be used for the leader. Default: font of the text line
<b>fontsize</b>	(FontSize) Size of the leader. Default: font size of the Textline
<b>text</b>	(Content string) The text which will be used for the leader. Default: U+002E ' (period)
<b>yposition</b>	(Float or keyword) Specifies the vertical position of the leader relative to the baseline as a numerical value or as one of the keywords fontsize, ascender, xheight, baseline, descender, textrise. Default: baseline

## 5.2 Single-Line Text with Textlines

*Cookbook* A full code sample can be found in the *Cookbook* topic `text_output/starter_textline`.

---

```
C++ Java void fit_textline(String text, double x, double y, String optlist)
Perl PHP fit_textline(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 placed on the page.

**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. See Section 6.1, «Object Fitting», page 109, for a description of the fitting algorithm.

**optlist** An option list specifying font, text, and formatting options. The following options are supported:

- ▶ General option: *errorpolicy* (see Section 2.5, «Exception Handling», page 27)
- ▶ Font loading options according to Table 4.3 for implicit font loading (i.e. *font* option in the text appearance group not supplied):  
*ascender, autocidfont, autosubsetting, capheight, descender, embedding, encoding, fallbackfonts, fontname, fontstyle, keepnative, linegap, metadata, monospace, readfeatures, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*
- ▶ Input filter options according to Table 5.1:  
*charref, escapesequence, glyphcheck, textformat*
- ▶ Text appearance options according to Table 5.2:  
*charspacing, dasharray, decorationabove, fakebold, fillcolor, font, fontsize, gstate, horzscaling, inittextstate, italicangle, kerning, overline, strikeout, strokecolor, strokewidth, textrendering, textrise, underline, underlineposition, underlinewidth, wordspacing*
- ▶ Shaping and typographic options according to Table 5.3:  
*features, language, script, shaping*
- ▶ Fitting options according to Table 6.1:  
*alignchar, blind, boxsize, fitmethod, margin, matchbox, orientate, position, rotate, stamp, showborder, shrinklimit*
- ▶ Options for Textline formatting according to Table 5.5:  
*inittextstate, leader, shadow, textpath, xadvancelist*

*Details* If *inittextstate=false* (which is the default), 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.

If *inittextstate=true* the default values of the text and graphics state parameters will be used to control the appearance of the text output unless they are explicitly overridden by options. The Textline options will not affect any output created after this call to `PDF_fit_textline()`.

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.

The reference point for *PDF\_continue\_text()* will not be set to the beginning of the text. In order to use *PDF\_continue\_text()* after *PDF\_fit\_textline()* you must query the starting point with *PDF\_info\_textline()* and the *startx/starty* keywords, and set the text position with *PDF\_set\_text\_pos()*.

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

*Params* See Table 4.8 and Table 4.9.

Table 5.5 Additional options for *PDF\_fit\_textline()*

<b>option</b>	<b>explanation</b>
<b>inittextstate</b>	(Boolean) If true all text appearance options will be initialized with the respective default values. If false the current text state values will be used. Default: false
<b>leader</b>	(Option list; will be ignored if <i>boxsize</i> is not specified or the width of the box is 0) Specifies filler text (e.g. dot leaders) and formatting options. Leaders will be inserted repeatedly between the border of the text box and the text. See Table 5.4 for a list of supported suboptions. Default: no leader
<b>shadow</b>	(Option list) Create a shadow effect for the text (default: no shadow): <b>offset</b> (List of 2 floats) The shadow's offset from the reference point of the text line in user coordinates or as a percentage of the font size. Default: {5% -5%} <b>fillcolor</b> (Color) Color of the shadow. Default: {gray 0.8} <b>gstate</b> (Gstate handle) Graphics state retrieved with <i>PDF_create_gstate()</i> which will be applied to the shadow. Default: none
<b>textpath</b>	(Option list) Draw text along a path. Text beyond the end of the path will not be displayed. See Table 5.6 for a list of supported suboptions. If <i>showborder=true</i> the flattened path will be drawn with the current linewidth and stroke color. The following options of <i>PDF_fit_textline()</i> have modified meaning for text on a path: <b>matchbox</b> A separate box will be created for each glyph. <b>position</b> The first value specifies the starting position of the text relative to the length of the path (left/center/right). If the text is longer than the path it will always begin at <i>startoffset</i> . The second value specifies the vertical position of each glyph relative to the path, i.e. which part of the glyph box will touch the path (bottom/center/top). <b>rotate</b> Specifies a rotation angle for each glyph. The following <i>fitbox</i> -related options will be ignored: <i>boxsize, margin, fitmethod, orientate, alignchar, showborder, stamp, leader</i> <i>Kerning and text with CJK legacy encodings are not supported for text on a path.</i>
<b>xadvancelist</b>	(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 <i>xadvance</i> values will be used instead of the standard glyph widths. Other effects, such as kerning and character spacing, are unaffected.

Table 5.6 Suboptions for the `textpath` option of `PDF_fit_textline()`

option	explanation
<b>path</b>	(Path handle; required) The path to use as baseline for text output. By default, the text will be placed at the left side of the path and the path will serve as the text baseline. However, if the second keyword in the position option is <code>top</code> the text will be placed to the right of the path and the top of the text will touch the path. The parameters <code>x</code> and <code>y</code> of <code>PDF_fit_textline()</code> will be used as reference point for the path.
<b>rotate</b>	(Float) Rotate the path, using the reference point as center and the specified value as rotation angle in degrees. Default: 0
<b>scale</b>	(List with one or two floats) Scale the path, using the reference point as center and the specified value(s) as horizontal and vertical scaling factor(s). If only one value is supplied it will be used for both directions. Default: {1 1}
<b>startoffset</b>	(Float or percentage) The offset of the starting point of the text along its path in user coordinates or as percentage of the path length. Default: 0
<b>tolerance</b>	(Float or percentage) Indicates how much the last glyph on the path is allowed to extend beyond the path. The value is specified in user coordinates or as a percentage of the fontsize. Default: 25%
<b>subpaths</b>	(List of integers or single keyword) List with the numbers of subpaths to be drawn. The keyword <code>all</code> specifies all subpaths. Default: <code>all</code>
<b>close</b>	(Boolean) If <code>true</code> , each subpath will be closed with a straight line. Default: the value specified when the path was constructed, or <code>false</code> if no value was specified
<b>round</b>	(Float) For each subpath, adjacent line vertices will be rounded in their joining point by a circular arc with the line segments as its tangents and with the specified radius. If the radius is negative the arc will be swept so that the corners are circularly grooved. If <code>close=true</code> and no line from the last to the first point was specified, the first line and the closing line will also be rounded. If <code>round=0</code> no rounding will be done. Default: the value specified when the path was constructed, or 0 if no value was specified

---

```

C++ Java double info_textline(String text, String keyword, String optlist)
Perl PHP float info_textline(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 without creating output 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 5.7.

**optlist** An option list specifying options for `PDF_fit_textline()`. Options which are not relevant for the requested keyword 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}.

If `errorpolicy=return` this function will return 0 in case of an error. If `errorpolicy=exception` this function will throw an exception in case of an error (even for the keyword `wellformed`).

**Scope** any except object

Table 5.7 Keywords for `PDF_info_textline()`

<b>keyword</b>	<b>explanation</b>
<b>angle</b>	Rotation angle of the baseline in degree, i.e. the text rotation
<b>ascender capheight descender</b>	Corresponding typographic value in user coordinates
<b>endx, endy</b>	x/y coordinates of the logical text end position in the user coordinate system
<b>height</b>	Height of the text string according to the boxheight specification of the matchbox
<b>perpendiculardir</b>	Unit vector perpendicular to writingdir; for standard horizontal text this would be (0, 1), for vertical text (1, 0)
<b>replacedchars</b>	Number of characters which have been replaced with a slightly different glyph from the internal list of typographically similar characters or with a glyph from a fallback font because they couldn't be mapped to a code in the current encoding or to a glyph in the font. This value can only be different from 0 if <code>glyphcheck=replace</code> .
<b>righttoleft</b>	1 if the global output direction for the text is right-to-left, and 0 for left-to-right or vertical text. The global direction will be determined based on the initial characters and any directional markers which may be present in the text (e.g. U+202D or &LRO; LEFT-TO-RIGHT OVERRIDE).
<b>scalex, scaley</b>	Horizontal and vertical scaling factors. If these are different from 1 the text had to be scaled to fit into the box.
<b>scriptlist</b>	String containing the space-separated list of the names of all scripts in the text. This may be useful to prepare text shaping. The script names are sorted by frequency in descending order. The scripts <code>_none</code> and <code>_latn</code> will be ignored since they are not relevant for shaping. If only <code>_none</code> and <code>_latn</code> characters are present in the text, -1 will be returned.
<b>startx, starty</b>	x/y coordinates of the logical text start position in the user coordinate system
<b>unknownchars</b>	If <code>glyphcheck=none</code> : number of skipped characters. The number includes character references which couldn't be resolved, and characters which couldn't be mapped to a code in the current encoding or to a glyph in the font. If <code>glyphcheck=replace</code> : number of characters which were replaced with the specified replacement character (option <code>replacementchar</code> ). The number includes characters which couldn't be mapped to a code in the current encoding or to a glyph in the font, and characters which couldn't be replaced with typographically similar characters.
<b>unmappedchars</b>	The number of characters which have been skipped or replaced, i.e. the sum of <code>replacedchars</code> and <code>unknownchars</code> .
<b>wellformed</b>	1 if the text is wellformed according to the font/encoding (and <code>textformat</code> , if applicable) selected in the corresponding options, otherwise 0.
<b>width</b>	Width of the text string (in horizontal writing mode) or width of the widest glyph (in vertical writing mode). Character spacing will not be applied after the last glyph.
<b>writingdirx writingdiry</b>	x/y coordinates of the dominant writing direction (direction of inline text progression), i.e. unit vector from (startx, starty) to (endx, endy). For left-to-right horizontal text the values will be (1, 0), for vertical text (0, -1), and for right-to-left horizontal text (-1, 0). The writing direction will be determined based on the shaping and vertical options as well as the directionality properties of the text.
<b>xheight</b>	Corresponding typographic value in user coordinates

## 5.3 Multi-Line Text with Textflows

*Cookbook* A full code sample can be found in the *Cookbook* topic `text_output/starter_textflow`.

---

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

**Perl PHP** `int add_textflow(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 (in PHP: 0) to create a new Textflow.

**text** (Content string) The contents of the Textflow. The text may not contain any in-line 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 5.3 and Table 5.8. The following options are supported:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ Font loading options according to Table 4.3 for implicit font loading (i.e. *font* option in the text appearance group not supplied):  
*ascender, autocidfont, autosubsetting, capheight, descender, embedding, encoding, fallbackfonts, fontname, fontstyle, keepnative, linegap, metadata, monospace, readfeatures, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, xheight*
- ▶ Input filter options according to Table 5.1:  
*charref, escapesequence, glyphcheck, textformat*
- ▶ Text appearance options according to Table 5.2:  
*charspacing, dasharray, decorationabove, fakebold, fillcolor, font, fontsize, gstate, horizscaling, inittextstate, italicangle, kerning, overline, strikeout, strokecolor, strokewidth, textrendering, textrise, underline, underlineposition, underlinewidth, wordspacing*
- ▶ Shaping and typographic options according to Table 5.3:  
*features, language, script, shaping*
- ▶ Options for Textflow formatting according to Table 5.8:  
*alignment, avoidemptybegin, fixedleading, hortabmethod, hortabsize, lastalignment, leader, leading, leftindent, minlinecount, parindent, rightindent, ruler, tabalignment*
- ▶ Options for controlling the line break algorithm according to Table 5.9:  
*adjustmethod, advancedlinebreak, avoidbreak, locale, maxspacing, minspacing, nofitlimit, shrinklimit, spreadlimit*
- ▶ Command options according to Table 5.10:  
*comment, mark, matchbox, nextline, nextparagraph, resetfont, return, space*
- ▶ Text semantics options according to Table 5.11:  
*charclass, charmapping, hyphenchar, tabalignchar*

**Returns** A Textflow handle which can be used in Textflow-related function calls. 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, 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 will preprocess the supplied text and options, but does not create any output in the generated PDF document, but only prepares the text. Use *PDF\_fit\_textflow()*, *PDF\_fit\_table()*, or *PDF\_fill\_textblock()* 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). These control characters will not be interpreted for symbolic fonts loaded with *encoding=builtin*. 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 5.8 Additional formatting options for *PDF\_add/create\_textflow()* and inline options in *PDF\_create\_textflow()*

option	explanation
<b>alignment</b>	(Keyword) Specifies formatting for lines in a paragraph. Default: left.
<b>left</b>	Left-aligned, starting at <i>leftindent+parindent</i> (for the first line of a paragraph) and at <i>leftindent</i> (for all other lines)
<b>center</b>	Centered between <i>leftindent</i> and <i>rightindent</i>
<b>right</b>	Right-aligned, ending at <i>rightindent</i>
<b>justify</b>	Left- and right-aligned
<b>avoid-emptybegin</b>	(Boolean) If true, empty lines at the beginning of a fitbox will be deleted. Default: false

Table 5.8 Additional formatting options for PDF\_add/create\_textflow() and inline options in PDF\_create\_textflow()

option	explanation
<b>fixedleading</b>	(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. fixedleading will be forced to true if the wrap option of PDF_fit_textflow() or the createwrapbox suboption of the matchbox option will be used to wrap the text around shapes. Default: false
<b>hortabmethod</b>	(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. <b>relative</b> The position will be advanced by the amount specified in hortabsize. <b>typewriter</b> The position will be advanced to the next multiple of hortabsize. <b>ruler</b> 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.
<b>hortabsize</b>	(Float or percentage) Width of a horizontal tab <sup>1</sup> . The interpretation depends on the hortabmethod option. Default: 7.5%
<b>lastalignment</b>	(Keyword) Formatting for the last line in a paragraph. All keywords of the alignment option are supported, plus the following (default: auto): <b>auto</b> Use the value of the alignment option if it is different from justify, else left
<b>leader</b>	(Option list) Specifies filler text (e.g. dot leaders) and formatting options. 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. See Table 5.4 for a list of supported suboptions. Default: no leader
<b>leading</b>	(Float or percentage) Distance between adjacent text baselines <sup>2</sup> . 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%
<b>leftindent</b>	(Float or percentage) Left indent of text lines <sup>1</sup> . 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
<b>minlinecount</b>	(Integer) Minimum number of lines in the last paragraph of 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
<b>parindent</b>	(Float or percentage) Left indent of the first line of a paragraph <sup>1</sup> . The amount will be added to leftindent. Specifying this option within a line will act like a tab. Default: 0
<b>rightindent</b>	(Float or percentage) Right indentation of text lines <sup>1</sup> . Default: 0
<b>ruler</b>	(List of floats or percentages) List of absolute tab positions for hortabmethod=ruler <sup>1</sup> . The list may contain up to 32 non-negative entries in ascending order. Default: integer multiples of hortabsize; if more than 32 horizontal tabs per line occur in the text the list will be extended with the corresponding default value.
<b>tabalignment</b>	(List of keywords; only with hortabmethod=ruler) Alignment for tab stops. The list may contain up to 32 entries. If more than 32 horizontal tabs per line occur in the text the list will be extended with the last value. Each entry in the list defines the alignment for the corresponding entry in the ruler option. Default: left. <b>center</b> Text will be centered at the tab position. <b>decimal</b> The first instance of tabalignchar will be left-aligned at the tab position. If no tabalignchar is found, right alignment will be used instead. <b>left</b> Text will be left-aligned at the tab position. <b>right</b> Text will be right-aligned at the tab position.

1. In user coordinates or as a percentage of the width of the fitbox

2. In user coordinates or as a percentage of the font size

Table 5.9 Additional options for controlling the line break algorithm for `PDF_add/create_textflow()` and inline options in `PDF_create_textflow()`

option	explanation
<b>adjustmethod</b>	(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 <code>minspacing</code> and <code>maxspacing</code> options. Default: <code>auto</code> .
<b>auto</b>	The following methods are applied in order: <code>shrink</code> , <code>spread</code> , <code>nofit</code> , <code>split</code> .
<b>clip</b>	Same as <code>nofit</code> , except that the long part at the right edge of the fitbox (taking into account the <code>rightindent</code> option) will be clipped.
<b>nofit</b>	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 <code>nofitlimit</code> option. Even justified paragraphs may look slightly ragged.
<b>shrink</b>	If a word doesn't fit in the line the text will be compressed subject to <code>shrinklimit</code> . If it still doesn't fit the <code>nofit</code> method will be applied.
<b>split</b>	The last word will not be moved to the next line, but will forcefully be split after the last character in the box. If <code>hyphenchar</code> is different from <code>none</code> a hyphen character will be inserted. Setting <code>hyphenchar=none</code> must be used to suppress the hyphen character (e.g. in formulae or URLs) since <code>PDFlib</code> does not automatically detect such situations.
<b>spread</b>	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 <code>spreadlimit</code> . If justification still cannot be achieved the <code>nofit</code> method will be applied.
<b>advanced-linebreak</b>	(Boolean) Enable the advanced line breaking algorithm which is required for complex scripts. This is required for linebreaking in scripts which do not use space characters for designating word boundaries, e.g. Thai. The options <code>locale</code> and <code>script</code> will be honored. Default: <code>false</code>
<b>avoidbreak</b>	(Boolean) If true, line breaking opportunities (e.g. at space characters) will be ignored until <code>avoidbreak</code> is reset to false. Mandatory line breaks (e.g. at a newline) and methods defined by <code>adjustmethod</code> will be still performed. In particular, <code>adjustmethod=split</code> may still create hyphenation. Default: <code>false</code>
<b>locale</b>	(Keyword) The locale which will be used for localized linebreaking methods if <code>advancedlinebreak=true</code> . The keywords consists of one or more components, where the optional components are separated by an underscore character '_' (the syntax slightly differs from NLS/POSIX locale IDs): <ul style="list-style-type: none"><li>▶ A required two- or three-letter lowercase language code according to ISO 639-2 (see <a href="http://www.loc.gov/standards/iso639-2">www.loc.gov/standards/iso639-2</a>), e.g. <code>en</code>, (English), <code>de</code> (German), <code>ja</code> (Japanese). This differs from the <code>language</code> option.</li><li>▶ An optional two-letter uppercase country code according to ISO 3166 (see <a href="http://www.iso.org/iso/country_codes/iso_3166_code_lists">www.iso.org/iso/country_codes/iso_3166_code_lists</a>), e.g. <code>DE</code> (Germany), <code>CH</code> (Switzerland), <code>GB</code> (United Kingdom)</li></ul> The keyword <code>_none</code> specifies that no locale-specific processing will be done. Specifying a locale is required for advanced line breaking for some scripts, e.g. Thai. Default: <code>_none</code> Examples: <code>tha</code> , <code>de_DE</code> , <code>en_US</code> , <code>en_GB</code>
<b>maxspacing</b> <b>minspacing</b>	(Float or percentage; only relevant if the line contains at least one space character <code>U+0020</code> and <code>alignment=justify</code> ) 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 <code>wordspacing</code> option will still be added). Defaults: <code>minspacing=50%</code> , <code>maxspacing=500%</code>
<b>nofitlimit</b>	(Float or percentage; only relevant with <code>alignment=justify</code> ) Lower limit for the length of a line with the <code>nofit</code> method <sup>1</sup> . Default: <code>75%</code> .
<b>shrinklimit</b>	(Percentage) Lower limit for compressing text with <code>adjustmethod=shrink</code> ; the calculated shrinking factor is limited by the provided value, but will be multiplied with the <code>horizscaling</code> option. Default: <code>85%</code>

Table 5.9 Additional options for controlling the line break algorithm for PDF\_add/create\_textflow() and inline options in PDF\_create\_textflow()

option	explanation
<b>spreadlimit</b>	(Float or percentage) Upper limit for the distance between characters for the spread method <sup>2</sup> ; the calculated distance will be added to the value of the charspacing option. Default: 0

Table 5.10 Additional command options for PDF\_add/create\_textflow() and inline options in PDF\_create\_textflow()

option	explanation
<b>comment</b>	(String) Arbitrary text which will be ignored; useful for commenting option lists or macros
<b>mark</b>	(Integer) Store the supplied number internally as a mark. The mark which has been stored most recently can later be retrieved with PDF_info_textflow() and the lastmark keyword. This may be useful for determining which portions of text have already been placed on the page.
<b>matchbox</b>	(Option list) Option list for creating a matchbox according to Table 6.3
<b>nextline</b> <b>nextparagraph</b>	(Boolean) Force a new line or paragraph.
<b>resetfont</b>	(Boolean) Reset font and fontsize to the most recently values which were different from the current settings (either different font or font size). 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, that differ from the first setting, and will be ignored otherwise.
<b>return</b>	(String; must not start with an underscore _ character) Exit PDF_fit_textflow() with the supplied string as return value.
<b>space</b>	(Float or percentage) The text position will be advanced by the provided value <sup>2</sup> .

Table 5.11 Additional text semantics options for PDF\_add/create\_textflow() and inline options in PDF\_create\_textflow()

option	explanation
<b>charclass</b>	(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 Unichars must be < 0xFFFF; will be ignored if advancedlinebreak=true) The specified Unichars will be classified by the specified keyword to determine the line breaking behavior of those character(s): <b>letter</b> behave like a letter (e.g. a B) <b>punct</b> behave like a punctuation character (e.g. + / ; : ) <b>open</b> behave like an open parenthesis (e.g. [ ) <b>close</b> behave like a close parenthesis (e.g. ] ) <b>default</b> reset all character classes to PDFlib's builtin defaults Example: charclass={ close » open « letter={ / : = } punct & }

Table 5.11 Additional text semantics options for `PDF_add/create_textflow()` and inline options in `PDF_create_textflow()`

option	explanation
<b>charmapping</b>	<p>(List of pairs, where each pair either contains two Unichars or a Unichar and a list of Unichar and integer; the Unichars must be &lt; 0xFFFF) 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:</p> <p><b>count &gt; 0</b> The replacement character will be repeated count times.</p> <p><b>count &lt; 0</b> A sequence of multiple instances of the character will be reduced to the absolute value of the specified number.</p> <p><b>count = 0</b> The character will be deleted.</p> <p>Examples:            charmapping={ hortab space CRLF space LF space CR space }            charmapping={ shy {shy 0} }            charmapping={ hortab {space 4} }</p>
<b>hyphenchar</b>	<p>(Unichar &lt; 0xFFFF 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</p>
<b>tabaligchar</b>	<p>(Unichar &lt; 0xFFFF) Character at which decimal tabs will be aligned. Default: U+002E ''</p>

**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 5.12. 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 5.12 Option list macro definitions and calls for `PDF_add/create_textflow()` and `PDF_fit_textflow()`

option	explanation
<b>macro</b>	(List of pairs) Each pair describes the name and definition of a macro as follows: <b>name</b> (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. <b>suboptlist</b> An option list which will literally replace the macro name when the macro is called. Leading and trailing whitespace will be ignored.
<b>&amp;name</b>	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 create_textflow(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 88), and inline option lists according to Table 5.8 and Table 5.13 (see also »Inline option lists for Textflows«, page 90). 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:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ All options of `PDF_add_textflow()` (see option list of `PDF_add_textflow()`)
- ▶ Options for controlling inline option list processing according to Table 5.13:  
*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. 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 90).

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 5.13 Additional options for inline option list processing in `PDF_create_textflow()`

option	explanation
<b>begoptlistchar</b>	(Unichar < 0xFFFF 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 90). If <code>textlen</code> is not specified, the <code>begoptlistchar</code> 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 <code>begoptlistchar</code> must be chosen such that it is contained in the encoding of the preceding text. The keyword <code>none</code> can be used to completely disable the search for option lists. Default: U+003C (<)
<b>endoptlistchar</b>	(Unichar < 0xFFFF ; U+007D '}' is not allowed) Character which terminates inline option lists. Default: U+003E (>)
<b>fixedtext-format</b>	(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 <code>optlist</code> parameter) If true, all text fragments and inline options lists will use the same <code>textformat</code> , which must be one of <code>utf8</code> , <code>utf16</code> , <code>utf16be</code> , or <code>utf16le</code> . 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 <code>textformat=bytes</code> , 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
<b>textlen</b>	(Integer or keyword; required for text fragments with <code>fixedtextformat=false</code> and <code>textformat=utf16xx</code> 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 90). The characters are counted before character references are resolved, e.g. <code>&lt;textlen=8&gt;&amp;#x2460;&lt;...&gt;</code> . The keyword <code>all</code> specifies all of the remaining text. Default: the text will be searched for the next occurrence of <code>begoptlistchar</code> .

**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 5.8. 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 (&#x3C; or &lt;) and start inline option lists with the literal < character:

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

Note that this method does not work for fonts with *encoding=builtin*.

- ▶ 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 fit_textflow(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.

**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. Shapes other than a rectangle can be filled with the *wrap* option.

**optlist** An option list specifying processing options according to Table 5.14. The following options can be used:

*blind, createfittext, createlastindent, exchangefillcolors, exchangestrokecolors, firstlinedist, fitmethod, fontscale, lastlinedist<sup>1</sup>, linespreadlimit, maxlines, minfontsize, orientate, returnatmark, rewind, rotate, showborder, showtabs, stamp, truncatetrailingwhitespace, verticalalign<sup>1</sup>, wrap*

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

- ▶ *\_stop*: all text in the Textflow has been processed. If the text was empty, *\_stop* will always be returned, even if the *return* or *mark/returnatmark* option was supplied.
- ▶ *\_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*: Some text was placed in the fitbox, but no more space is available, or the maximum number of lines (as specified via the *maxlines* option) has been placed in

the fitbox, or *fitmethod=auto* and *minfontsize* has been specified but the text didn't fit into the fitbox. 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 fitbox is too small to hold any text, or a wrapbox was larger than the fitbox. No more calls to *PDF\_fit\_textflow()* with the same fitbox should be issued in order to avoid infinite loops.
- ▶ *\_mark#*: The option *returnatmark* has been specified with the number #, and the mark with the number specified in this option has been placed.
- ▶ 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 text appearance options (see Table 5.2) 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 5.14 Options for *PDF\_fit\_textflow()*

<b>option</b>	<b>explanation</b>
<b>blind</b>	(Boolean) If <i>true</i> , no output will be generated, but all calculations will be performed and the formatting results can be checked with <i>PDF_info_textflow()</i> . Default: <i>false</i>
<b>createfittext</b>	(Boolean) If <i>true</i> the text placed in the current fitbox will be saved in memory so that it can later be retrieved with a call to <i>PDF_info_textflow()</i> and the keyword <i>fittext</i> . Default: <i>true</i>
<b>createlast-indent</b>	(Option list) Reserve some space at the end of the last line in the fitbox and optionally create a matchbox which can be used to fill the reserved space. The reserved space may be useful to add continuation dots, an image, a link to the continuation of the text, etc. at the end of the text. Supported suboptions: <b>rightindent</b> (Float or percentage) Additional right indent of the last text line in the fitbox in user coordinates or as percentage of the width of the fitbox. The value will be added to the value of the <i>rightindent</i> option of <i>PDF_add/create_textflow()</i> . Default: <i>0</i> <b>matchbox</b> (Option list according to Table 6.3) Create a matchbox at the end of the last line. If the <i>matchbox</i> option <i>boxwidth</i> is not specified, the value of <i>rightindent</i> will be used as <i>boxwidth</i> . If <i>boxwidth=0</i> no box will be created.
<b>exchange-fillcolors</b>	(List with an even number of colors) Each pair in the list specifies an original fill color and a replacement color. Whenever the <i>Textflow</i> specifies the original fill color within the fitbox it will be replaced with the specified replacement color. This may be useful to adjust the colors to the background. Example: <i>exchangefillcolors={{gray 0} white Orchid DeepPink {rgb 1 0 1} MediumBlue}</i>
<b>exchange-strokecolors</b>	(List with an even number of colors) Each pair in the list specifies an original stroke color and a replacement color. Whenever the <i>Textflow</i> specifies the original stroke color within the fitbox it will be replaced with the specified replacement color. This may be useful to adjust the colors to the background.

Table 5.14 Options for PDF\_fit\_textflow()

option	explanation
<b>firstlinedist<sup>1</sup></b>	<p>(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 <code>fixedleading=true</code>, and the maximum of all font sizes in the line otherwise), or as a keyword. Default: <code>leading</code>.</p> <p><b>leading</b> The leading value determined for the first line; typical diacritical characters such as <code>À</code> will touch the top of the fitbox.</p> <p><b>ascender</b> The ascender value determined for the first line; typical characters with larger ascenders, such as <code>d</code> and <code>h</code> will touch the top of the fitbox.</p> <p><b>capheight</b> The capheight value determined for the first line; typical capital uppercase characters such as <code>H</code> will touch the top of the fitbox.</p> <p><b>xheight</b> The xheight value determined for the first line; typical lowercase characters such as <code>x</code> will touch the top of the fitbox.</p> <p>If <code>fixedleading=false</code> the maximum of all <code>leading</code>, <code>ascender</code>, <code>xheight</code>, or <code>capheight</code> values found in the first line will be used.</p>
<b>fitmethod</b>	<p>(Keyword) Specifies the method used to fit the text into the fitbox. Default: <code>clip</code></p> <p><b>auto</b> PDF_fit_textflow() will repeatedly be called in blind mode with reduced font size and other font-related options (see <code>fontscale</code>) until the text fits into the fitbox (but see also option <code>minfontsize</code>)</p> <p><b>clip</b> The text will be truncated at the bottom of the fitbox.</p> <p><b>nofit</b> The text can extend beyond the bottom of the fitbox.</p>
<b>fontscale</b>	<p>(Float or percentage) Values of <code>fontsize</code> and absolute values (but not percentages) of <code>leading</code>, <code>minspacing</code>, <code>maxspacing</code>, <code>spreadlimit</code>, and <code>space</code> will be multiplied with the supplied scaling factor or percentage. Default: 1 if <code>rewind=0</code>, otherwise the value supplied with the corresponding call to <code>PDF_fit_textflow()</code>.</p>
<b>gstate</b>	<p>(Gstate handle) Handle for a graphics state retrieved with <code>PDF_create_gstate()</code>. The graphics state affects all text placed with this function. If another graphics state has already been supplied to <code>PDF_add/create_textflow()</code> both graphics states will be merged. Default: no graphics state (i.e. current settings will be used)</p>
<b>lastlinedist<sup>1</sup></b>	<p>(Float, percentage, or keyword; will be ignored for <code>fitmethod=nofit</code>) 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 <code>fixedleading=true</code>, 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:</p> <p><b>descender</b> The descender value determined for the last line; typical characters with descenders, such as <code>g</code> and <code>j</code> will touch the bottom of the fitbox.</p> <p>If <code>fixedleading=false</code> the maximum of all descender values found in the last line will be used.</p>
<b>linespread-limit</b>	<p>(Float or percentage; only for <code>verticalalign=justify</code>) Maximum amount in user coordinates or as percentage of the leading for increasing the leading for vertical justification. Default: 200%</p>
<b>maxlines</b>	<p>(Integer or keyword) Maximum number of lines in the fitbox, or the keyword <code>auto</code> which means that as many lines as possible will be placed in the fitbox. When the maximum number of lines has been placed <code>PDF_fit_textflow()</code> will return the string <code>_boxfull</code>. Default: <code>auto</code></p>
<b>minfontsize</b>	<p>(Float or percentage) Minimum font size allowed when text is scaled down to fit into the fitbox, especially for <code>fitmethod=auto</code>. The limit is specified in user coordinates or as a percentage of the height of the fitbox. If the limit is reached and the text still does not fit the string <code>_boxfull</code> will be returned. Default: 0.1%</p>
<b>mingapwidth</b>	<p>(Float or percentage) Minimal horizontal width for fitting text between shapes (e.g. between wrap contours) in user coordinates or as a percentage of the <code>fontsize</code>. This may be useful to avoid ugly formatting results in cases where only small gaps are left between wrap contours. Default: 10%</p>

Table 5.14 Options for PDF\_fit\_textflow()

option	explanation
<b>orientate</b>	(Keyword) Specifies the desired orientation of the text when it is placed. Default: north. <b>north</b> upright <b>east</b> pointing to the right <b>south</b> upside down <b>west</b> pointing to the left
<b>returnatmark</b>	(Integer) PDF_fit_textflow() will return prematurely at the text position where the Textflow option mark is defined with the specified number. The return reason string will be _mark#, where # is the number specified in this option.
<b>rewind</b>	(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. <b>1</b> Rewind to the state before the first call to PDF_fit_textflow(). <b>0</b> Don't reset the Textflow. <b>-1</b> Rewind to the state before the last call to PDF_fit_textflow(). <b>-2</b> Rewind to the state before the second last call to PDF_fit_textflow().
<b>rotate</b>	(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 fitbox and the text being rotated. The rotation will be reset when the text has been placed. Default: 0
<b>showborder</b>	(Boolean) If true, the border of the fitbox and all wrap boxes will be stroked (using the current graphics state). This may be useful for development and debugging. Default: false
<b>showtabs</b>	(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): <b>none</b> no lines will be drawn <b>fitbox</b> lines will be drawn over the full height of the fitbox <b>validarea</b> lines will be drawn only in vertical area where they are valid
<b>stamp</b>	(Keyword) This option can be used to create a diagonal stamp within the fitbox. Line breaks for the stamp text should be specified explicitly (i.e. with newline characters or the newline option). If the text does not contain any explicit line breaks a single-line stamp will be created. The generated stamp text will be as large as possible, but not larger than the specified fontsize. Supported keywords (default: none): <b>llzur</b> The stamp will run diagonally from the lower left corner to the upper right corner. <b>ulzlr</b> The stamp will run diagonally from the upper left corner to the lower right corner. <b>none</b> No stamp will be created.
<b>truncate-trailing-whitespace</b>	(Boolean) Control treatment of fitboxes which contain only trailing whitespace, i.e. the text in the fitbox starts with whitespace and there is only whitespace until the end of the Textflow. If this option is true, trailing whitespace is removed, i.e. the fitbox is treated as empty and the return value is _stop. If this option is false, the whitespace is processed like regular text, i.e. the function may return a value different from _stop (depending on the amount of trailing whitespace) and the textendx/y and other keywords of PDF_info_textflow() take the whitespace into account. truncate-trailing-whitespace=false may be useful if the original text must be processed without any whitespace removal. Default: true

Table 5.14 Options for PDF\_fit\_textflow()

option	explanation
<b>verticalalign</b> <sup>1</sup>	(Keyword) Vertical alignment of the text in the fitbox; the firstlinedist and lastlinedist options will be taken into account as appropriate (default: top): <ul style="list-style-type: none"> <li><b>top</b> 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.</li> <li><b>center</b> 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.</li> <li><b>bottom</b> 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.</li> <li><b>justify</b> 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. If this limit is exceeded no justification will be performed. The height of the first line will only be increased if firstlinedist=leading.</li> </ul>
<b>wrap</b>	(Option list according to Table 5.15) The text will run around the areas specified with the suboptions listed in Table 5.15. This can be used to place images or paths within the Textflow and wrap the text around it, or to fill arbitrary shapes with text. The fitbox will be filled according to the fillrule option, starting at the border of the fitbox. <p>By default, the specified areas will not contain any text (except where they overlap), i.e. the text is wrapped around the shapes. Using the addfitbox and inversefill options the opposite effect can be achieved: the specified areas will be filled with text, and the rest of the fitbox remains empty. This can be used to fill arbitrary shapes (and not only the rectangle supplied in the llx/lly/urx/ury parameters) with text.</p> <p>Absolute and relative coordinate values will be interpreted in the user coordinate system. A relative coordinate will be added to the previous absolute coordinate. Up to 256 values can be supplied as relative values. Percentages will be interpreted in the fitbox coordinate system, i.e. the lower left corner of the fitbox is (0, 0) and the upper right corner is (100, 100) (even in a topdown system). Up to 256 values can be supplied as percentage. Examples:</p> <p>Exclude a box with relative coordinates: wrap={ boxes={{120r 340r 50r 60r}} }</p> <p>(equivalent to wrap={ boxes={{120 340 170 400}} }</p> <p>Exclude the upper right quarter of the fitbox: wrap={ boxes={{50% 50% 100% 100%}} }</p> <p>Fill a triangular shape: wrap={ addfitbox polygons={{50% 80% 30% 40% 70% 40% 50% 80%}} }</p> <p>Exclude the area of an image with a matchbox called image1: wrap={ usematchboxes={{ image1 }}} }</p>

1. The firstlinedist, lastlinedist and verticalalign options always refer to the fitbox, even in the presence of wrap elements. This means – especially in the case of inverse filling, i.e. the wrap elements are filled with text – that Textflow will not use the bounding box of the wrap elements to determine the distance between text and fitbox borders and the position of the text box according to the verticalalign option. This may lead to unexpected results, especially if the outer edges of the wrap elements don't touch the fitbox. This effect can almost completely be avoided by supplying wrap elements which touch the fitbox.

**C++ Java** `double info_textflow(int textflow, String keyword)`

**Perl PHP** `float info_textflow(int textflow, string keyword)`

**C** `double PDF_info_textflow(PDF *p, int textflow, const char *keyword)`

Query the current state of a Textflow after a call to PDF\_fit\_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 5.16.

**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 5.15 Suboptions for the wrap option of PDF\_fit\_textflow()

option	explanation
<b>addfitbox</b>	(Boolean) If true, the fitbox will be added to the wrap area. As a result, the shapes specified with other wrapping options will be filled with text instead of wrapping the text around the shapes. Default: false
<b>beziers</b>	(List of two or more Bézier curves) Two or more Bézier curves which will be added to the wrap area.
<b>boxes</b>	(List of rectangles) One or more rectangles which will be added to the wrap area.
<b>circles</b>	(List of circles) One or more circles which will be added to the wrap area.
<b>creatematchboxes</b>	(List of option lists) Create matchboxes from one or more rectangles in the boxes option. Each option list corresponds to one entry in the boxes option (ordering is relevant), and controls the creation of a matchbox. All relevant matchbox options in Table 6.3 can be used. A suboption list can be empty; in this case no matchbox will be created for the corresponding wrap box.
<b>fillrule</b>	(Keyword) Specifies the method for determining the interior of overlapping wrap shapes (default: even-odd). See Table 7.2 for details: <b>evenodd</b> Use the even-odd rule. <b>winding</b> Use the non-zero winding number rule. Use this rule to process the interior of overlapping circles (i.e. to avoid »doughnut holes«), or to process the union of overlapping shapes (instead of the intersection).
<b>inversefill</b>	(Boolean) If true, wrap shape processing starts at the first intersection of the text line with the border of a wrap element inside the fitbox. If false, processing starts at the fitbox border. If fillrule=evenodd, the option inversefill=true has the same effect as addfitbox=true. If fillrule=winding, the option addfitbox=true leads to an empty or a full fitbox (for inversefill=false or true, respectively).
<b>lineheight</b>	(List with two elements, each being a positive float or a keyword) Defines the vertical extent of the text line to be used for calculating the intersection with wrap areas. Two keywords/floats specify the extent above and below the text baseline. Supported keywords: none (no extent), xheight, descender, capheight, ascender, fontsize, leading, textrise Default: {ascender descender}
<b>usematchboxes</b>	(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 added to the wrap area.
<b>offset</b>	(Float or percentage) Horizontal distance between the text and the contour of the wrap area, supplied in user coordinates or as a percentage of the width of the fitbox. This can be used to horizontally extend the wrap area. Default: 0
<b>paths</b>	(List of option lists) One or more path objects which will be added to the wrap area. Supported suboptions: <b>path</b> (Path handle; required) Handle for the path to be added to the wrap area. <b>refpoint</b> (List of two floats or percentages) Coordinates of the reference point for the path in user coordinates or as percentages of the width and height of the fitbox. Default: {0 0} The following options of PDF_draw_path() can also be used (see Table 6.1 and Table 7.8): align, attachmentpoint, boxsize, close, fitmethod, orientate, position, round, scale, subpaths
<b>polygons</b>	(List of polylines) One or more polylines (not necessarily closed) which will be added to the wrap area.

Table 5.16 Keywords for `PDF_info_textflow()`

<b>keyword</b>	<b>explanation</b>
<b>boundingbox</b>	Handle of the path containing the Textflow's bounding box in user coordinates or -1 (0 in PHP)
<b>boxlinecount</b>	Number of lines in the last fitbox
<b>firstparalinecount</b>	Number of lines in the first paragraph of the fitbox
<b>firstlinedist</b>	Distance between the first text baseline and the fictitious baseline above (if <code>verticalalign=top</code> this will be the upper border of the fitbox)
<b>fittest</b>	String index for a text string which corresponds to the text placed in the previous call to <code>PDF_fit_textflow()</code> . This can be used to determine the amount of text which could be placed in the fitbox. The string will be normalized as follows: encoding is UTF-16 in Unicode-capable languages or (EBCDIC-)UTF-8 otherwise, line breaks will be marked with <code>U+000A</code> , and horizontal tabs will be replaced with a space character <code>U+0020</code> .
<b>fontscale</b>	The value of <code>fontscale</code> after the most recent call to <code>PDF_fit_textflow()</code> with <code>fitmethod=auto</code> .
<b>lastfont</b>	Handle of the font used in the last text line in the fitbox
<b>lastfontsize</b>	Font size used in the last text line in the fitbox
<b>lastmark</b>	Number of the last mark found in the processed part of the Textflow in the last fitbox (marks can be set with the <code>mark</code> option)
<b>lastlinedist</b>	Distance between the last text baseline and the fictitious baseline below, assuming unmodified leading (if <code>verticalalign=bottom</code> this will be the lower border of the fitbox)
<b>lastparalinecount</b>	Number of lines in the last paragraph of the fitbox
<b>leading</b>	The current value of the <code>leading</code> option, as determined by the text and options within the Textflow
<b>leftlinex<sup>1</sup>, leftliney<sup>1</sup></b>	The x and y coordinates of the line with the leftmost start in the most recently filled fitbox, in current user coordinates
<b>maxlinelength</b>	Length of the longest text line in the most recently filled fitbox
<b>maxliney<sup>1</sup></b>	The y coordinate of the baseline of the longest text line in the most recently filled fitbox, in current user coordinates
<b>minlinelength</b>	Length of the shortest text line in the most recently filled fitbox
<b>minliney<sup>1</sup></b>	The y coordinate of the baseline of the shortest text line in the most recently filled fitbox, in current user coordinates
<b>returnreason</b>	String index which can be used with the <code>string</code> parameter in <code>PDF_get_parameter()</code> (see Table 2.3) to retrieve the return reason of the most recent direct or indirect call to <code>PDF_fit_textflow()</code> . The retrieved return reason will be one of the return strings of <code>PDF_fit_textflow()</code> . This is useful for querying the result of indirect Textflow calls issued internally by <code>PDF_fill_textblock()</code> .
<b>rightlinex<sup>1</sup>, rightliney<sup>1</sup></b>	The x and y coordinates of the line with the rightmost end in the most recently filled fitbox, in current user coordinates
<b>split</b>	Specifies whether word splitting occurred in the last fitbox: <b>0</b> No word had to be split. <b>1</b> At least one word had to be split.
<b>textendx, textendy</b>	The x or y coordinate of the current text position after the most recently filled fitbox in current user coordinates
<b>textheight</b>	Height of the bounding box of the whole text (taking <code>firstlinedist</code> and <code>lastlinedist</code> into account) in current user coordinates

Table 5.16 Keywords for `PDF_info_textflow()`

keyword	explanation
<code>textwidth</code>	Width of the bounding box of the whole text in current user coordinates
<code>used</code>	Percentage of text (0...100) which has been placed so far
<code>x1, y1, ..., x4, y4</code>	Coordinates of the bounding box of the whole text (taking <code>firstlinedist</code> and <code>lastlinedist</code> into account) in current user coordinates

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

---

**C++ Java** `void delete_textflow(int textflow)`  
**Perl PHP** `delete_textflow(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

## 5.4 Table Formatting

*Cookbook* A full code sample can be found in the *Cookbook* topic `tables/starter_table`.

---

```
C++ Java int add_table_cell(int table, int column, int row, string text, string optlist)
Perl PHP int add_table_cell(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) to start a new table. 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 5.17. The following options can be used:

- ▶ General option: `errorpolicy` (see Section 2.5, »Exception Handling«, page 27)
- ▶ Column and row definition: `colwidth`, `colscalegroup`, `minrowheight`, `return`, `rowheight`, `rowjoingroup`, `rowscalegroup`
- ▶ Cell properties: `checkwordsplitting`, `colspan`, `margin`, `marginleft`, `marginbottom`, `marginright`, `margintop`, `matchbox`, `rowspan`
- ▶ Cell contents: `annotationtype`, `continuetextflow`, `fieldname`, `fieldtype`, `fitannotation`, `fitfield`, `fitimage`, `fitpdipage`, `fittextline`, `fittextflow`, `fitpath`, `image`, `path`, `pdipage`, `repeat-content`, `textflow`

**Returns** A table handle which can be used in subsequent table-related calls. If `errorpolicy=return` the caller must check for a return value of -1 (in PHP: 0) since it 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, path objects, form fields, annotations, Textflows, or Textlines. Multiple content types can be specified for a particular cell in a single function call.

See the PDFlib Tutorial for a description of the table formatting algorithm and width and height calculations.

**Scope** any except *object*

Table 5.17 Options for PDF\_add\_table\_cell()

key	explanation
<b>annotation-type</b>	(String) Specifies the type of an annotation to be inserted in the table cell according to Table 12.6.
<b>checkword-splitting</b>	(Boolean; only relevant for Textflow cells) If true, the table formatter will check whether the Textflow requires at least one forced word splitting when fitting the text into the table cell. If so, the cell width will be increased in an attempt to avoid word splittings. Default: true
<b>colscale-group<sup>1</sup></b>	(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.
<b>colspan</b>	(Integer) Number of columns spanned by the cell. Default: 1
<b>colwidth<sup>1</sup></b>	(Float or percentage) Width of the column specified in the column parameter. The width can be specified in user coordinates <sup>2</sup> , or as a percentage of the width of the table's first fitbox (see PDF_fit_table()). User coordinates and percentages must not be mixed, i.e. either user coordinates or percentages must be used in all column width definitions of a table. The column width may be increased automatically if the column traverses cells containing text. Images and PDF pages in table cells don't have any influence on column widths. Default: see option colwidthdefault of PDF_fit_table()
<b>continue-textflow</b>	(Boolean; only relevant for Textflows) If true the contents of the Textflow specified in the textflow option can be continued in another cell provided that the other cell is filled with the same Textflow handle and continuetextflow=true as well. The parts of the Textflow will be placed in the order in which the cells are added. PDFlib will not adjust the cell size to the whole Textflow, and the checkwordsplitting option will be ignored. Therefore, a suitable cell size should be defined. If false the Textflow will be started from the beginning. Default: false
<b>fieldname</b>	(Hypertext string) Form field name for fieldtype.
<b>fieldtype</b>	(String) Specifies the type of a form field to be inserted in the table cell according to Table 12.9. Form field groups should be defined outside of tables.
<b>fitannotation</b>	(Option list) Annotation options for annotationtype according to Table 12.7.
<b>fitfield</b>	(Option list) Form field options for fieldtype according to Table 12.10.
<b>fitimage</b>	(Option list; only relevant for images and templates) Option list for PDF_fit_image(). This option list will be applied to place the image or template supplied in the image option 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. <sup>3</sup>
<b>fitpath</b>	(Option list; only relevant for path objects) Option list for PDF_draw_path(). This option list will be applied to place the path object specified in the path option within its bounding box in the cell. The lower left corner of the inner cell box will be used as 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. <sup>3</sup>
<b>fitpdi<sup>4</sup>page</b>	(Option list; only relevant for PDI pages; only if PDI is available) 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. <sup>3</sup>



Table 5.17 Options for `PDF_add_table_cell()`

key	explanation
<b>rowjoin-group<sup>1</sup></b>	(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.
<b>rowspan</b>	(Integer) Number of rows spanned by the cell. Default: 1
<b>textflow</b>	(Textflow handle) The Textflow associated with the handle will be placed in the inner cell box. The <code>continuetextflow</code> option controls the behavior for a Textflow handle which is used in multiple cells. The Textflow handle must not be used outside the table. Default: no Textflow

1. The last specification of this option is dominant; earlier specifications for the same row or column will be ignored.
2. More precisely, the coordinate system which is in effect when `PDF_fit_table()` is called for placing the first table instance.
3. The box size will be calculated automatically; the `boxsize` option in the supplied option list will be ignored.

---

**C++ Java** `String fit_table(int table, double llx, double lly, double urx, double ury, String optlist)`  
**Perl PHP** `string fit_table(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 5.18. The following options can be used:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ Fitting options according to Table 6.1: *fitmethod, position, showborder*
- ▶ General table options: *blind, colwidthdefault, horshrinklimit, rewind, rowheightdefault, vertshrinklimit*
- ▶ Table contents: *header, footer*
- ▶ Table decoration: *fill, firstdraw, gstate, stroke*
- ▶ Visualization aids for development and debugging: *debugshow, 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()`.

The error behavior can be changed with the *errorpolicy* parameter or option.

**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 in the fitbox, the first table instance will be placed; more table instances can be placed with subsequent calls to this function de-

pending on the return value. The contents of a table cell will be placed in the following order:

- ▶ Filling: 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 filling: 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, path objects, Textflow, Textline, annotations, form fields.
- ▶ Matchbox ruling: single cell areas which are defined by a *matchbox* definition.
- ▶ Ruling: the lines specified with the *stroke* option will be stroked according to the *linecap* and *linejoin* suboptions of the *stroke* option in the following order: *other, horother, hor#, horlast, vertother, vert#, vertlast, frame* (the order of horizontal and vertical lines can be changed with the *firstdraw* option). Cells which span multiple rows or columns will not be intersected 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* Generally *page, pattern, template, glyph*; however, if the table contains form fields or annotations the respective scopes are dominant. For example, a table containing form fields or annotations cannot be placed on a template.

Table 5.18 Options for `PDF_fit_table()`

key	explanation
<b>blind</b>	(Boolean) If <code>true</code> , all calculations will be performed, but no output will be created. The formatting results can be checked with <code>PDF_info_table()</code> . Default: <code>false</code>
<b>colwidth-default</b>	(Float or keyword; only relevant in the first call to <code>PDF_fit_table()</code> for a particular table) Default width for columns which do not contain any <code>Textline</code> nor <code>Textflow</code> and for which the <code>colwidth</code> option of <code>PDF_add_table_cell()</code> was not specified. The default width can be specified as an absolute value or as a keyword. The value 0 (zero) is equivalent to the keyword <code>distribute</code> . The following keywords are supported (default: <code>auto</code> ): <ul style="list-style-type: none"> <li><b>auto</b> Columns with unspecified width which contain only <code>Textline</code> cells will have the width of the text. The remaining width of the fitbox will be distributed among all rows with <code>Textflow</code> or other cells. The table covers the full width of the fitbox.</li> <li><b>distribute</b> The width of the fitbox will be distributed equally among all columns with unspecified width and which don't contain any <code>Textline</code>. The table covers the full width of the fitbox unless it contains only <code>Textlines</code>.</li> <li><b>minimum</b> Columns with unspecified width which contain only <code>Textline</code> cells will have the width of the text, i.e. the smallest possible width to hold the text.</li> </ul> In order to create columns with minimal width you can supply a small value (e.g. 1). The width of all columns which contain <code>Textline</code> or <code>Textflow</code> cells will be adjusted automatically (see <code>PDFlib Tutorial</code> ).
<b>debugshow</b>	(Boolean) If <code>true</code> , all errors for tables which are too high, too wide, or where the cells get too small will be suppressed and logged instead. The resulting table instance will be created as a debugging aid although the table is damaged. Default: <code>false</code>

Table 5.18 Options for PDF\_fit\_table()

key	explanation
<b>fill</b>	<p>(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 6.2, »Matchboxes«, page 115):</p> <p><b>area</b> (Keyword) Table area(s) to be filled:</p> <p><b>col#</b> column number # in the table</p> <p><b>collast</b> last column</p> <p><b>coleven</b> all columns with even numbers (according to col in PDF_add_table_cell())</p> <p><b>colodd</b> all columns with odd numbers</p> <p><b>colother</b> all unspecified columns</p> <p><b>row#</b> row number # in the table</p> <p><b>rowlast</b> last body row in the table instance</p> <p><b>roweven</b> all rows with even numbers (according to row in PDF_add_table_cell())</p> <p><b>rowodd</b> all rows with odd numbers</p> <p><b>header</b> all rows in the header group</p> <p><b>footer</b> all rows in the footer group</p> <p><b>rowother</b> all unspecified body rows</p> <p><b>table</b> complete table area (i.e. all rows in the table)</p> <p>The following graphics appearance options according to Table 7.2 can also be used:  <b>fillcolor, shading</b></p> <p>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}} }            Use fillcolor=none to suppress shading for a table area.</p>
<b>firstdraw</b>	<p>(Keyword) Specifies the order in which horizontal and vertical lines will be created (default: vertlines):</p> <p><b>horlines</b> Horizontal lines will be created first.</p> <p><b>vertlines</b> Vertical lines will be created first.</p>
<b>footer</b>	<p>(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)</p>
<b>gstate</b>	<p>(Gstate handle) Handle for a graphics state retrieved with PDF_create_gstate(). All table decorations will be subject to the supplied graphics state. The cell contents will not be affected. Default: no gstate (i.e. current settings will be used).</p>
<b>header</b>	<p>(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)</p>
<b>horshrinklimit</b>	<p>(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%</p>
<b>rewind</b>	<p>(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):</p> <p><b>1</b> Rewind to the state before the first call to PDF_fit_table().</p> <p><b>0</b> Don't reset the table.</p> <p><b>-1</b> Rewind to the state before the last call to PDF_fit_table() (the one before the current call)</p>

Table 5.18 Options for PDF\_fit\_table()

key	explanation
<b>rowheight-default</b>	<p>(Float or keyword; only relevant in the first call to PDF_fit_table() for a particular table) Default height for rows for which the rowheight option of PDF_add_table_cell() was not specified. The default height can be specified as an absolute value or as a keyword. If a float value is specified it will be used as default row height unless it is smaller than the textbox height. The value 0 (zero) is equivalent to the keyword distribute. The following keywords are supported (default: auto):</p> <p><b>auto</b> Rows which contain only Textline cells will have a height of two times the height of the textbox. The remaining height of the fitbox will be distributed among all rows with Textflow or other cells. The table covers the full height of the fitbox.</p> <p><b>distribute</b> The height of the fitbox will be distributed equally among all rows with unspecified height. The table covers the full height of the fitbox.</p> <p><b>minimum</b> Rows with unspecified height which contain only Textline cells will have the height of the textbox, i.e. the smallest possible height to hold the text. You can use the boxsize or margin options to increase the height of Textline cells.</p> <p>In order to create rows with minimal height you can supply a small positive value (e.g. 1). The height of all rows which contain Textline or Textflow cells will be adjusted automatically (see PDFlib Tutorial).</p>
<b>showcells</b>	(Boolean) If true, the border of each inner cell box will be stroked using the current graphics state. Default: false
<b>showgrid</b>	(Boolean) If true, the vertical and horizontal boundary of all columns and rows will be stroked. Default: false
<b>stroke</b>	<p>(List of option lists) This option can be used to create stroked lines at the cell borders:</p> <p><b>line</b> (Keyword) Table line(s) to be stroked:</p> <p><b>vert#</b> vertical line at the right border of column number #; vert0 is the left table border</p> <p><b>vertfirst</b> first vertical line (equivalent to vert0)</p> <p><b>vertlast</b> last vertical line</p> <p><b>vertother</b> all unspecified vertical lines</p> <p><b>hor#</b> horizontal line at the bottom of row number # in the table; row0 is the top border</p> <p><b>horfirst</b> first horizontal line in the table instance</p> <p><b>horother</b> all unspecified horizontal lines</p> <p><b>horlast</b> last horizontal line in the table instance</p> <p><b>frame</b> outer border of the table</p> <p><b>other</b> all unspecified lines</p> <p>The following graphics appearance options according to Table 7.2 can also be used:  <b>dasharray, dashphase, linecap, linejoin, linewidth, strokecolor</b></p> <p>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} }  Use strokecolor=none to suppress stroking for a table area.</p>
<b>vertshrink-limit</b>	(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 info_table(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 5.19.

**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 5.19 Keywords for `PDF_info_table()`

<b>keyword</b>	<b>explanation</b>
<b>boundingbox</b>	Handle of the path containing the table instance's bounding box in user coordinates or -1 (0 in PHP)
<b>firstbodyrow</b>	Number of the first body row in the most recently placed table instance
<b>height</b>	Height of the table instance
<b>horboxgap</b>	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).
<b>horshrinking</b>	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, otherwise it will be 100.
<b>lastbodyrow</b>	Number of the last body row in the most recently placed table instance
<b>returnreason</b>	String index of the return reason
<b>rowcount</b>	Number of rows in the most recently placed table instance (including headers and footers)
<b>rowsplit</b>	1 if the last row had to be split, 0 otherwise
<b>vertboxgap</b>	Difference between the height of the most recently generated 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).
<b>vert-shrinking</b>	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, otherwise it will be 100.
<b>width</b>	Width of the table instance
<b>x1, y1, ... , x4, y4</b>	Coordinates of the corners of the table instance in user coordinates, counterclockwise starting at the lower left corner
<b>xvrtline#</b>	x coordinate of the vertical line with number #. xvrtline0 is the left table border.
<b>yhorline#</b>	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 delete_table(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 5.20.

*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 5.20 Options for *PDF\_delete\_table()*

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



# 6 Object Fitting and Matchboxes

## 6.1 Object Fitting

PDFlib's fitting algorithm places a rectangular graphical object relative to a point, a horizontal or vertical line, or a rectangle. The fitting algorithm is implemented in several functions:

- ▶ *PDF\_fit\_textline()*, *PDF\_info\_textline()*
- ▶ *PDF\_fit\_image()*, *PDF\_info\_image()*
- ▶ *PDF\_fit\_pdi\_page()*, *PDF\_info\_pdi\_page()*
- ▶ *PDF\_draw\_path()*, *PDF\_info\_path()*
- ▶ *PDF\_add\_table\_cell()* (via option lists for the *fittextline*, *fitimage*, *fitpdipage*, *fitpath* options)
- ▶ *PDF\_fit\_table()*
- ▶ *PDF\_fill\_\*block()*

*Note* Since the fitting options for *Textflow* are slightly different they are not described here, but in Section 5.3, »Multi-Line Text with *Textflows*«, page 83.

Table 6.1 lists fitting options which can be supplied to the fitting functions. Not all options are available for all functions, and the behavior of some options may slightly change depending on the function; see Table 6.1 for details. The following options form the group of fitting options:

*alignchar*, *boxsize*, *dpi*, *fitmethod*, *margin*, *matchbox*, *minfontsize*, *orientate*, *position*, *refpoint*, *rotate*, *scale*, *stamp*, *showborder*, *shrinklimit*

**Object box.** In all cases the fitting algorithm calculates the smallest enclosing rectangle of the placed object. This rectangle is called the *object box*. It can be modified according to the type of object:

- ▶ Textlines (*PDF\_fit/info\_textline()*, single-line text Blocks, table cells): the default width of the object box is the width of the text, and the default height of the text box is the *capheight* of the selected font. This can be changed with the *boxheight* suboption of the *matchbox* option.
- ▶ Images and templates (*PDF\_fit/info\_image()*, image Blocks, table cells): the suboption *clipping* of the *matchbox* option can be used to define some part of the object as object box. For TIFF and JPEG images with a clipping path the smallest enclosing rectangle with edges parallel to the coordinate axes will be used as object box if the suboption *innerbox* of the *matchbox* option is set.
- ▶ Imported PDF pages (*PDF\_fit/info\_pdi\_page()*, PDF Blocks, table cells): the suboption *clipping* of the *matchbox* option can be used to define some part of the object as object box.
- ▶ Path objects (*PDF\_draw/info\_path()*, table cells): the smallest rectangle with edges parallel to the coordinate axes which encloses the path will be used as object box. The object box will only be calculated if the *boxsize* and *position* options have values different from zero.
- ▶ Table instances (*PDF\_fit\_table()*): the smallest rectangle with edges parallel to the coordinate axes which encloses the table instance will be used as object box.

**Reference point.** The *reference point* is used as an anchor for placing the object box. It is defined as follows:

- ▶ In *PDF\_fit\_\**() and *PDF\_draw\_path*(): the *x* and *y* function parameters;
- ▶ In *PDF\_info\_\**(): the point (*o*, *o*); *PDF\_info\_path*() additionally supports the *refpoint* option for specifying the reference point.
- ▶ *PDF\_add\_table\_cell*(), *PDF\_fit\_table*(), and *PDF\_fill\_\*block*(): the lower left corner of the table cell, table instance, or PDFlib Block; *PDF\_fill\_\*block*() additionally supports the *refpoint* option for specifying the reference point.

**Fitbox and reference line segment.** The rectangle in which the object box will be placed is called the *fitbox*. It has the reference point (*x*, *y*) as its lower left corner and its size is specified by the two values of the *boxsize* option:

```
lower left corner = (x, y)
upper right corner = x + boxsize[0], y + boxsize[1]    (if topdown=false)
upper right corner = x + boxsize[0], y - boxsize[1]    (if topdown=true)
```

In addition to the definition above the fitbox can be modified as follows:

- ▶ Textlines: the fitbox can be reduced with the *margin* option;
- ▶ table cells: the fitbox is defined by the inner cell box, i.e. the cell box as modified by the *margin\** options;
- ▶ table instances: the fitbox is defined by the *llx/lly/ury/ury* parameters;
- ▶ PDFlib Blocks: the fitbox is by default defined by the Block's *Rect* property, but it can be modified with the *refpoint* and/or *boxsize* options.

In the last three cases above the fitbox is always available; otherwise it is only available if the *boxsize* option was specified with two values different from zero.

If *boxsize[0]=0* the box degenerates to a vertical line. The fitting algorithm will place the object box relative to this line segment. Similarly, if *boxsize[1]=0* the box will be placed relative to the resulting horizontal line segment. The vertical or horizontal line segment is called the *reference line segment*.

**Placing the object box.** The object box can be placed in different ways:

- ▶ If no fitbox is available the object will be placed relative to the reference point (not for table cells, table instances, and PDFlib Blocks): the lower left corner of the object box will coincide with the reference point. Using the *position* option other points within the object box can be selected. For example, *position=center* places the object box's center point at the reference point.  
The option *scale* will be honored for images, templates, path objects, and imported PDF pages; the option *dpi* will be honored for images. The *fitmethod* option will be ignored in this case.  
Path objects: if *position={o o}* the bounding box will not be calculated and the origin of the path object will coincide with the reference point.
- ▶ Relative to a reference line segment (not for table cells, table instances, and PDFlib Blocks): this works similarly to placing an object relative to the reference point as described above. In addition, the *position* option also defines a point on the line segment which will serve as reference point.
- ▶ Relative to the fitbox: The *fitmethod* option specifies whether and how the object box will be forced to fit into the fit box. If *fitmethod=nofit* nothing will be done to restrict the result to the fitbox. Other values of *fitmethod* define details of the fitting algo-

rithm according to Table 6.2.

In this case the options *scale* and *dpi* will be ignored, and the options *margin*, *shrinklimit*, and *showborder* will be honored.

The lower left corner of the object box will coincide with the lower left corner of the fitbox. Using the *position* option other points within the object box and simultaneously the corresponding point within the fitbox can be selected. For example, *position=center* places the object box's center point at the center point of the fitbox.

Table 6.1 Fitting options for various functions

option	explanation
<b>align</b>	(List of two floats; only for path objects) The coordinates of a direction vector in user coordinates which defines the rotation of the path object. The x direction of the path object's coordinate system will be aligned with the specified vector. The coordinates must not both be 0. The calculated rotation will be added to the rotation defined by the orientate option. Default: {1 0}, i.e. no additional rotation
<b>alignchar</b>	(Unichar < 0xFFFF or keyword; only for Textlines) 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.  If this option is present the formatted text may exceed beyond the fitbox. This option will be ignored if the specified alignment character is not present in the text. If the specified character cannot be found in the font or encoding, an exception will be thrown if glyphcheck=error. For other values of glyphcheck the alignchar option will silently be ignored if the character is not available.  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
<b>attachment-point</b>	(String; only for path objects) Name of the attachment point: If fitmethod=nofit the path object will be placed so that the specified attachment point coincides with the reference point. Default: origin of the path object
<b>blind</b>	(Boolean) If true, no output will be generated, but all calculations will be performed and the formatting results can be checked with the appropriate info function PDF_info_*( ). Default: false
<b>boxsize</b>	(List of floats; not for tables) Width and height of the fitbox, relative to which the object (possibly rotated according to the rotate option) will be placed. The lower left corner of the fitbox coincides with the reference point (x, y). Placing the object 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 fitmethod option will be ignored and the object will be placed relative to the vertical line from (x, y) to (x, y+height) (or (x, y-height) for topdown systems), or the horizontal line from (x, y) to (x+width, y), according to the position option.  Default for Blocks: width and height of the Block's Rect property Default for all other fitting functions: {0 0}
<b>dpi</b>	(List of floats; only for images) One or two values specifying the desired image resolution in pixels per inch in horizontal and vertical direction. This option does not change the number of pixels in the image (downsampling). If a single value is supplied it will be used for both dimensions. With the value zero the image's internal resolution will be used if available, or 72 dpi otherwise. The keyword internal is equivalent to zero. The scaling resulting from this option is relative to the current user coordinate system; if the coordinate system has been scaled the resulting physical resolution will be different from the supplied values. The scale option will be applied in addition to the dpi values.  This option will be ignored if the fitmethod option has been supplied with one of the keywords auto, meet, slice, or entire. Default: internal
<b>fitmethod</b>	(Keyword) Method used to fit the object into the specified fitbox. See Table 6.2 for supported keywords. Keywords other than nofit will be ignored if no fitbox has been specified.  Default: cLip for Textflow; meet for tables, path objects and reference option; and nofit otherwise

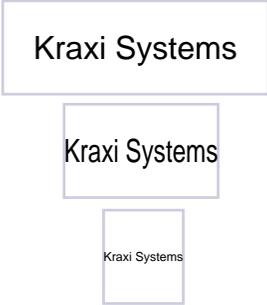
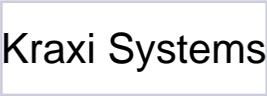
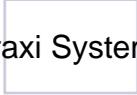
Table 6.1 Fitting options for various functions

<b>option</b>	<b>explanation</b>
<b>margin</b>	(List of floats; only for <code>Textlines</code> ) One or two float values describing additional horizontal and vertical reductions of the fitbox. Default: 0
<b>matchbox</b>	(Option list; not for path objects) Option list for creating a matchbox according to Table 6.3
<b>minfontsize</b>	(Float or percentage; only for <code>Textflow</code> ) Minimum allowed font size when text is scaled down to fit into the fitbox with <code>fitmethod=auto</code> when <code>shrinklimit</code> is exceeded. The limit is specified in user coordinates or as a percentage of the height of the fitbox. If the limit is reached the text will be created with the specified <code>minfontsize</code> as <code>fontsize</code> . Default: 0.1%
<b>orientate</b>	<p>(Keyword or float; not for tables) Specifies the desired orientation of the object relative to the current coordinate system. Default: north.</p> <p>Arbitrary rotation angles (in degrees) can be specified for path objects, but not other object types. The bounding box of the path object will be calculated after rotating the path object. All functions support the following keywords (corresponding equivalent angles are shown in parentheses):</p> <p><b>north</b>     upright (0)</p> <p><b>east</b>        pointing to the right (270)</p> <p><b>south</b>      upside down (180)</p> <p><b>west</b>        pointing to the left (90)</p>
<b>position</b>	<p>(List of floats or keywords) One or two values specifying the position of the object box relative to the reference point, the reference line segment, or the fitbox. The values specify a position within the object box. This position is defined horizontally as percentage of the box width (first value) and vertically as percentage of the box height (second value). This specified position coincides with the reference point, a point on the reference line segment or a point within the fitbox. Although the values designate percentages, they must be specified without any percent sign. Negative values are allowed. If both values are equal, it is sufficient to specify a single value. Default: {0 100} for tables, center for the reference option, otherwise {0 0}</p> <p>Examples:</p> <p>{0 0}     The lower left corner of the object box coincides with the reference point, the start of the reference line segment, or the lower left corner of the fitbox.</p> <p>{100 100}     The upper right corner of the object box coincides with the reference point, the end of the reference line segment, or the upper right corner of the fitbox.</p> <p>The keywords <code>left</code>, <code>center</code>, <code>right</code> (in x direction) or <code>bottom</code>, <code>center</code>, <code>top</code> (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. Examples:</p> <p>{left center}     or {0 50}     left-aligned</p> <p>{center}            or {50 50}     centered</p> <p>{right center}     or {100 50}     right-aligned</p> <p>Only for <code>Textlines</code>: the keyword <code>auto</code> can be used for the first value in the list. It indicates <code>right</code> if the writing direction of the text is from right to left (e.g. for Arabic and Hebrew text), and <code>left</code> otherwise (e.g. for Latin text).</p>
<b>repoint</b>	<p>(List of floats; only for <code>PDF_fill_*block()</code> and <code>PDF_info_path()</code>) Specifies the reference point in user coordinates for fitting the block contents or path.</p> <p>Default for <code>PDF_fill_*block()</code>: lower left corner of the rectangle defined by the Block's <code>Rect</code> property</p> <p>Default for <code>PDF_info_path()</code>: {0 0}</p>
<b>rotate</b>	(Float; not for tables, table cells, path objects) Rotate the coordinate system, using the reference point as center and the specified value as rotation angle in degrees. This results in the fitbox and the object being rotated. The rotation will be reset when the object has been placed. Default: 0

Table 6.1 Fitting options for various functions

option	explanation
<b>scale</b>	(List of floats; only for images, templates, imported PDF pages, path objects) Scales the object in horizontal and vertical direction by the specified scaling factors (not percentages), using the reference point as center. If both factors are equal it is sufficient to specify a single value. This option will be ignored if the <code>fitmethod</code> option has been supplied with one of the keywords <code>auto</code> , <code>meet</code> , <code>slice</code> , or <code>entire</code> . Default: <code>{1 1}</code>
<b>stamp</b>	(Keyword; only for Textlines; will be ignored if <code>boxsize</code> is not specified) This option can be used to create a diagonal stamp of maximal size in the rectangle specified with the <code>boxsize</code> option. More specifically, the text will be placed diagonally in the fitbox. The size of the text box will be chosen so that it covers the fitbox as much as possible while preserving the aspect ratio of the text box (i.e. the text comprising the stamp will be as large as possible). The options <code>fontsize</code> , <code>fitmethod</code> , and <code>position</code> will be ignored. The options <code>orientate=west</code> and <code>=east</code> don't make any sense (only north and south). Supported keywords (default: none): <b>llzur</b> The stamp runs diagonally from the lower left corner to the upper right corner. <b>ulzlr</b> The stamp runs diagonally from the upper left corner to the lower right corner. <b>none</b> No stamp will be created.
<b>showborder</b>	(Boolean) If <code>true</code> , the border of the fitbox will be stroked using the current graphics state. If a stamp is created, the bounding box of the stamp will also be stroked. This may be useful for development and debugging. Default: <code>false</code>
<b>shrinklimit</b>	(Float or percentage; only for Textlines) The lower limit of the shrinkage factor which will be applied to fit text with <code>fitmethod=auto</code> . Default: <code>0.75</code>

Table 6.2 Keywords for the fitmethod option of various functions; the illustrations demonstrate the typical effect of each keyword on a Textline, using the same value for the fontsize option in all examples.

keyword	explanation	
<b>auto</b>	<p>(Only for Textlines; other object types: same behavior as meet) This method tries to fit the text box into the fitbox automatically.</p> <p>In detail: Same as nofit if the text fits into the fitbox. Otherwise a scaling factor is calculated such that the text will be shrunk horizontally (distorted) to fit into the fitbox. If the calculated factor is smaller than the shrinklimit option, the meet method is applied by reducing the fontsize until the text can be fit or the value of minfontsize is reached.</p>	
<b>clip</b>	<p>Position the object and graphically clip it at the edges of the fitbox.</p> <p>PDF_fit_table(): the calculated table box will be logically clipped at the bottom edge of the fitbox and can be continued in the next fitbox. Logical clipping is similar to PDF_fit_textflow(), but not graphical clipping as in PDF_fit_image() etc. The table box will be placed inside the fitbox according to the position option.</p>	
<b>entire</b>	<p>Scale the object box such that it entirely covers the fitbox. Generally this method will distort the object. The position option doesn't have any effect.</p> <p>PDF_fit_table(): similar to clip. If the table box is smaller than the fitbox, the cells of the table box (but not their contents) will be enlarged uniformly until the table box entirely covers the fitbox.</p>	
<b>meet</b>	<p>Position the object according to the position option, and scale it such that it entirely fits into the fitbox while preserving its aspect ratio. Generally at least two edges of the object box will meet the corresponding edges of the fitbox.</p> <p>PDF_fit_table(): similar to clip. If the table box is smaller than the fitbox, the cells of the table box (but not their contents) will be enlarged uniformly until the horizontal or vertical table edge meets the fitbox.</p>	
<b>nofit</b>	<p>Position the object only. The scale and dpi will be applied to images.</p> <p>PDF_fit_table(): The table will be calculated for a virtual fitbox with infinite height. The table box will be placed inside the fitbox according to the position option. The default sizes of columns and rows relate to the specified fitbox height. fitmethod=nofit is recommended to format the table in blind mode.</p>	
<b>slice</b>	<p>Position the object according to the position option, and scale it such that it entirely covers the fitbox, while preserving the aspect ratio and making sure that at least one dimension of the object is fully contained in the fitbox. Generally parts of the object's other dimension will extend beyond the fitbox, and will therefore be clipped.</p> <p>PDF_fit_table(): similar to clip. If the table box is smaller than the fitbox the cells of the table box (but not their contents) will be enlarged uniformly until the fitbox is entirely covered by the table box while preserving its aspect ratio. The table box will be placed inside the fitbox according to the position option. The parts of the table box which exceed beyond the fitbox will be clipped graphically at the edges of the fitbox.</p>	

## 6.2 Matchboxes

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

- ▶ Textlines: *PDF\_fit\_textline()*, *PDF\_fill\_textblock()* with *textflow=false*
- ▶ Textflows: *PDF\_add/create\_textflow()*, *PDF\_fill\_textblock()* with *textflow=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 following suboptions:

- ▶ Graphics appearance options according to Table 7.2:  
*borderwidth, dasharray, dashphase, fillcolor, gstate, linecap, linejoin, shading, strokecolor*
- ▶ Matchbox controlling options according to Table 6.3

Details of the rectangle(s) corresponding to a matchbox can be queried with *PDF\_info\_matchbox()*.

Table 6.3 Suboptions for the *matchbox* option of various functions

option	explanation
<b>boxheight</b>	(List with two elements, each being a positive float or a keyword; only for Textline and Textflow) Vertical extent of the text box. Two values can be specified numerically or via keywords for the extent above and below the baseline: none (no extent), xheight, descender, capheight, ascender, fontsize, leading, textrise With Textflows the values corresponding to the text at the beginning of the matchbox will be used. Default: {capheight none}
<b>boxwidth</b>	(Float or percentage; only for Textflow) Width of the matchbox specified in user coordinates or as a percentage of the box height. 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
<b>clipping</b>	(Rectangle 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%}
<b>create-wrapbox</b>	(Boolean; only for Textflow) If true, the rectangle(s) comprising the matchbox will be inserted as wrap areas in the Textflow after they have been calculated. The subsequent lines after the lines containing the matchbox will be wrapped around the rectangle(s). Default: false
<b>doubleadapt</b>	If true the start and end point of the second line will be adapted to the first line. Otherwise the second line will be shorter or longer by the amount of doubleoffset. Default: true
<b>doubleoffset</b>	(Float) If different from 0 the lines around the border of the inner matchbox rectangle will be doubled. The second line has the specified offset from the original line. If the offset is positive the line will be drawn outside the matchbox rectangle, and inside if the offset is negative. Default: 0 (i.e. single line)
<b>drawleft</b> <b>drawbottom</b> <b>drawright</b> <b>drawtop</b>	(Boolean) If true, the corresponding border of the rectangle will be drawn provided that the borderwidth is set to a value greater than 0. Default: true

Table 6.3 Suboptions for the matchbox option of various functions

option	explanation
<b>end</b>	(Boolean; only for Textflow) Specifies the end of the matchbox. If true, all other suboptions for the current matchbox definition 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.
<b>exceedlimit</b>	(Float or percentage; only for Textflow) Upper limit for the part of the matchbox which is allowed to exceed beyond the bottom or right edge of the fitbox, specified in user coordinates or as a percentage of the matchbox height. If the specified limit would be exceeded PDF_fit_textflow() will return _boxfull; the remaining text and the matchbox can be continued in the next fitbox. Default: 0, i.e. the matchbox must completely fit into the box.
<b>innerbox</b>	(Boolean; only for table cells, and TIFF and JPEG images) Table cells: If true, the cell box will be reduced by the margins defined for the cell; otherwise the full cell box will be used. TIFF and JPEG images: If true and the image contains a clipping path the bounding box of the clipping path will be used instead of the full image. Default: false
<b>margin</b>	(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%). This option will be ignored for an edge for which offset* has been supplied. Default: 0
<b>name</b>	(Name string) Name of the matchbox. If the name has already been assigned to a matchbox, another rectangle for this name will be created. This means that a matchbox may consist of more than one rectangle. The name can be used in PDF_info_matchbox(). Various functions support the option usematchbox to reference one or more rectangles of a matchbox, e.g. to add an annotation with PDF_create_annotation(). Matchbox names can be used until the end of the current page. Default: no name
<b>offsetleft</b> <b>offsetbottom</b> <b>offsetright</b> <b>offsettop</b>	(Float or percentage) User-defined offset from the left/right/bottom/top edge of the calculated 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
<b>openrect</b>	(Boolean; only for Textflow and table cells) Textflow: If true and 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 true and 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
<b>round</b>	(Float) Adjacent lines of a matchbox rectangle will be joined with a circular arc with the specified radius and the line segments as tangents. If the specified radius is negative the arc segments will be swept inwards, and the tangents will be perpendicular to the line segments of the box. Default: 0 (no rounding)

---

```

C++ Java double info_matchbox(String boxname, int num, String keyword)
Perl PHP float info_matchbox(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.

Alternatively, the name '\*' (single asterisk character) can be used to enumerate all matchboxes on the page.

**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). See Table 6.4 for the special case *num*=0.

**keyword** A keyword specifying the requested information according to Table 6.4.

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

**Details** Named matchboxes within a Textflow can only be queried after calling *PDF\_fit\_textflow()*.

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

Table 6.4 Keywords for *PDF\_info\_matchbox()*

<b>keyword</b>	<b>explanation</b>
<b>count</b>	<i>(The num parameter will be ignored)</i> If <i>boxname</i> contains the name of a matchbox: Number of rectangles for this matchbox on the page If <i>boxname</i> =*: number of matchboxes with at least one rectangle on the page
<b>exists</b>	If <i>boxname</i> contains the name of a matchbox: 1 if the rectangle exists, 0 otherwise. With <i>boxname</i> =* this keyword can be used to enumerate all matchboxes on the page: if <i>num</i> =0: 1 if a matchbox exists at all, 0 otherwise otherwise: 1 if a matchbox with number <i>num</i> exists
<b>height</b> <sup>1</sup>	Height of the rectangle in user coordinates
<b>name</b>	String index for the name of the matchbox with number <i>num</i> . The corresponding string can be retrieved via <i>PDF_get_parameter()</i> and the <i>string</i> parameter (see Table 2.3).
<b>rectangle</b>	Handle of the path containing the <i>num</i> -th rectangle in user coordinates or -1 (in PHP: 0)
<b>width</b> <sup>1</sup>	Width of the rectangle in user coordinates
<b>x1, y1, ... , x4, y4</b> <sup>1</sup>	Position of the <i>i</i> -th rectangle corner ( <i>i</i> =1, 2, 3, 4) in user coordinates. In the coordinate system of the respective fit element (image, text, etc.), <i>x1</i> , <i>y1</i> correspond to the lower left, <i>x2</i> , <i>y2</i> to the lower right, <i>x3</i> , <i>y3</i> to the upper right and <i>x4</i> , <i>y4</i> to the upper left corner.

1. This keyword will be ignored if *boxname*=\*



# 7 Graphics Functions

Cookbook A full code sample can be found in the Cookbook topic `graphics/starter_graphics`.

## 7.1 Graphics Appearance Parameters and Options

Table 7.1 lists relevant parameter key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 7.1 Path-related keys for `PDF_get/set_parameter()`

key	explanation
<code>fillrule</code>	Set the current fill rule to winding or evenodd (see Table 7.2). Scope: page, pattern, template, glyph

Table 7.2 lists graphics appearance options for `PDF_create_gstate()`, `PDF_draw_path()`, `PDF_shading_pattern()`, the `fill` and `stroke` options of `PDF_fit_table()`, and the `matchbox` option of various functions.

Table 7.2 Graphics appearance options

option	explanation and possible values
<code>borderwidth</code>	(Float; only for matchboxes) Line width for the rectangle's border. If you set <code>borderwidth</code> to a value greater than 0 all rectangle borders will be stroked. To prevent the upper, lower, left, or right border from being stroked, set the corresponding <code>drawtop</code> , <code>drawbottom</code> , <code>drawleft</code> , or <code>drawright</code> option to false. Default: 0
<code>dasharray</code>	(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.
<code>dashphase</code>	(Float) Distance into the dash pattern at which to start the dash. Default: 0
<code>fillcolor</code>	(Color) Fill color of the area. Default: generally {gray 0} (in PDF/A mode: {lab 0 0 0}), but none for tables and matchboxes
<code>fillrule</code>	(Keyword) Fill rule which determines the interior of areas for filling and clipping (default: winding): <ul style="list-style-type: none"> <li><b>winding</b> Use the nonzero winding number rule. For simple shapes, the result of filling matches intuitive expectations. For shapes consisting of multiple paths the direction of the paths is relevant.</li> <li><b>evenodd</b> Use the even-odd rule, which yields the same results as winding for simple shapes, but produces different results for more complex shapes, especially self-intersecting paths.</li> </ul>
<code>flatness</code>	(Float > 0) A positive number which describes the maximum distance (in device pixels) between a circular arc or a curve and an approximation constructed from straight line segments. Default: 1
<code>gstate</code>	(Gstate handle) Handle for a graphics state retrieved with <code>PDF_create_gstate()</code> . Default: no graphics state (i.e. current settings will be used)

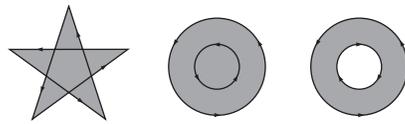


Table 7.2 Graphics appearance options

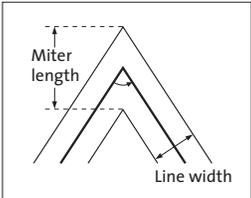
option	explanation and possible values	
<b>linecap</b>	(Integer or keyword) Shape at the end of a path (default: butt):	
	<b>butt</b> (Equivalent value: 0) Butt end caps: the stroke is squared off at the endpoint of the path.	
	<b>round</b> (Equivalent value: 1) Round end caps: a semicircular arc with a diameter equal to the line width is drawn around the endpoint and filled in.	
	<b>projecting</b> (Equivalent value: 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.	
<b>linejoin</b>	(Integer or keyword) Shape at the corners of paths (default: miter):	
	<b>miter</b> (Equivalent value: 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.	
	<b>round</b> (Equivalent value: 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.	
	<b>bevel</b> (Equivalent value: 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.	
<b>linewidth</b>	(Float > 0) Line width. Default: 1	
<b>miterlimit</b>	(Float >= 1) Controls the spike produced by miter joins (default: 10; this corresponds to an angle of roughly 11.5 degrees)	
	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 linewidth exceeds the miter limit.	
<b>shading</b>	(Option list according to Table 7.3; only for matchboxes and tables) Specify a shading for the matchbox's rectangle(s) or table area, using the value of the fillcolor option (if specified) or the current fill color as the starting color.	
<b>strokecolor</b>	(Color) Stroke color of the path. Default: generally {gray 0} (in PDF/A mode: {lab 0 0 0}), but none for tables and matchboxes	

Table 7.3 Suboptions for the shading graphics appearance option

<b>option</b>	<b>explanation</b>
<b>antialias</b>	(Boolean) Specifies whether to activate antialiasing for the shading. Default: false
<b>domain</b>	(List of 2 Floats) Two numbers specifying the limiting values of a parametric variable $t$ . The variable is considered to vary linearly between these two values as the color gradient varies between the starting and ending points of the axis. Default: {0 1}
<b>end</b>	(List of 2 floats or percentages) The $x$ and $y$ coordinates of the end point for the shading axis (type=axial) or a point on the circle to calculate the radius (type=radial), specified as percentages of the rectangle's width and height or in user coordinates. Default: {100% 100%}
<b>endcolor</b>	(Color; required) Color for the end point.
<b>N</b>	(Float) Exponent for the color transition function; must be $> 0$ . Default: 1
<b>start</b>	(List of 2 floats or percentages) The $x$ and $y$ coordinates of the starting point for the shading axis (type=axial) or the center of the shading circle (type=radial), specified as percentages of the rectangle's width and height or in user coordinates. Default: {0% 0%}
<b>type</b>	(Keyword) Shading type: axial (linear shading) or radial (radial shading). Default: axial

## 7.2 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 4, »Font and Text Functions«, page 51.

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

---

**C++ Java** `void setdash(double b, double w)`  
**Perl PHP** `setdash(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** `setdashpattern(string optlist)`  
**C** `void PDF_setdashpattern(PDF *p, const char *optlist)`

---

Set a dash pattern defined by an option list.

**optlist** Graphics appearance options according to Table 7.2 (an empty list will generate a solid line): *dasharray, dashphase*

*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** `setflat(float flatness)`  
**C** `void PDF_setflat(PDF *p, double flatness)`

---

Set the *flatness* tolerance.

**flatness** The *flatness* tolerance, see Table 7.2.

*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** `setlinejoin(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 7.2. The *linejoin* style must be specified as one of the numbers 0, 1, or 2.

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

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

---

**C++ Java** `void setlinecap(int linecap)`

**Perl PHP** `setlinecap(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 7.2. The *linecap* parameter must be specified as one of the numbers 0, 1, or 2.

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

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

---

**C++ Java** `void setmiterlimit(double miter)`

**Perl PHP** `setmiterlimit(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, see Table 7.2.

**Details** 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** `setlinewidth(float width)`

**C** `void PDF_setlinewidth(PDF *p, double width)`

---

Set the current line width.

**width** The linewidth in units of the 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** `initgraphics()`

**C** `void PDF_initgraphics(PDF *p)`

---

Reset all color and graphics state parameters to their default values

**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 default values. 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*

---

**C++ Java** `void save()`

**Perl PHP** `save()`

**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()`;
- ▶ text position and the following text-related parameters: *charspacing*, *decorationabove*, *fakebold*, *font*, *fontsize*, *horizscaling*, *italicangle*, *leading*, *strokewidth*, *textrendering*, *textrise*, *underlineposition*, *underlinewidth*, *wordspacing*.

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** `restore()`

**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.

---

**C++ Java** `int create_gstate(String optlist)`

**Perl PHP** `int create_gstate(string optlist)`

**C** `int PDF_create_gstate(PDF *p, const char *optlist)`

---

Create a graphics state object subject to various options.

**optlist** An options list with graphics state options:

- ▶ Graphics appearance options according to Table 7.2:  
*flatness, linecap, linejoin, linewidth, miterlimit*
- ▶ Graphics state options according to Table 7.4:  
*alphaishape, blendmode, opacitystroke, overprintfill, overprintmode, overprintstroke, renderingintent, smoothness, softmask, 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 7.4 Options for `PDF_create_gstate()`

key	explanation and possible values
<b>alphaishape</b>	(Boolean; PDF 1.4) Sources of alpha are treated as shape (true) or opacity (false). Default: false
<b>blendmode</b>	(Keyword list; PDF 1.4; <i>f</i> used with PDF/X-1, PDF/X-3, or PDF/A-1 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
<b>opacityfill</b>	(Float; PDF 1.4; if used in PDF/A mode it must have the value 1) Opacity for fill operations in the range 0..1. The value 0 means fully transparent; 1 means fully opaque.
<b>opacitystroke</b>	(Float; PDF 1.4; if used in PDF/A mode it must have the value 1) Opacity for stroke operations in the range 0..1. The value 0 means fully transparent; 1 means fully opaque.
<b>overprintfill</b>	(Boolean) Overprint for operations other than stroke. Default: false
<b>overprintmode</b>	(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

Table 7.4 Options for `PDF_create_gstate()`

key	explanation and possible values
<b>overprintstroke</b>	(Boolean) Overprint for stroke operations. Default: false
<b>renderingintent</b>	(Keyword) Color rendering intent used for gamut compression; possible keywords: Auto, AbsoluteColorimetric, RelativeColorimetric, Saturation, Perceptual
<b>smoothness</b>	(Float) Maximum error of a linear interpolation for a shading; must be $\geq 0$ and $\leq 1$
<b>softmask</b>	(Option list or keyword) Current soft mask with mask shape or opacity values for transparent imaging. The keyword none specifies no soft mask at all; this is required to disable soft masks which may be in effect from a previously set gstate. Supported options (default: none): <b>backdropcolor</b> (List with one, three, or four floats; only relevant for type=luminosity) Color to be used as the backdrop against which to composite the transparency group template. The number of float values depends on the colorspace suboption of the transparencygroup option used when creating the transparency group template (1 for DeviceGray, 3 for DeviceRGB, 4 for DeviceCMYK). Default: black in the respective colorspace <b>template</b> (Template handle; required) Transparency group template which has been created with <code>PDF_begin_template_ext()</code> and the <code>transparencygroup</code> option. <b>type</b> (Keyword; required) Method for deriving mask values from the transparency group template: <b>alpha</b> Use the transparency group's alpha value and ignore the color. <b>luminosity</b> Convert the transparency group's color to a single-component luminosity value.
<b>strokeadjust</b>	(Boolean) Whether or not to apply automatic stroke adjustment. Default: false
<b>textknockout</b>	(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** `set_gstate(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*

## 7.3 Coordinate System Transformations

All transformation functions (*PDF\_translate()*, *PDF\_scale()*, *PDF\_rotate()*, *PDF\_align()*, *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.

---

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

**Perl PHP** *translate(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** *scale(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** *rotate(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 align(double dx, double dy)`

**Perl PHP** `align(float dx, float dy)`

**C** `void PDF_align(PDF *p, double dx, double dy)`

---

Align the coordinate system with a relative vector.

**dx, dy** Coordinates of a direction vector *dx* and *dy* must not both be 0.

**Details** Rotate the coordinate system such that the *x* axis of the new coordinate system is aligned with the vector (*dx*, *dy*), and the *y* axis is aligned with (-*dy*, *dx*). This is equivalent to `PDF_rotate()` with  $\text{phi}=180^\circ / \text{pi} * \text{atan2}(\text{dy}/\text{dx})$ .

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

---

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

**Perl PHP** `skew(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  $-360^\circ < \text{alpha}, \text{beta} < 360^\circ$ , and must be different from  $-270^\circ, -90^\circ, 90^\circ, \text{ and } 270^\circ$ .

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

---

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

**Perl PHP** `concat(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** `setmatrix(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*

## 7.4 Path Construction

Table 7.5 lists relevant value key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19). The key names cannot be used with `PDF_set_value()` since there are corresponding API functions available for setting these values.

Table 7.5 Keys for `PDF_get_value()`

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

*Note* Make sure to call one of the functions in Section 7.5, »Painting and Clipping«, page 134, 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** `moveto(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** `lineto(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** `curveto(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  $(x_3, y_3)$ , using  $(x_1, y_1)$  and  $(x_2, y_2)$  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** `circle(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** `arc(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** *arcn(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 circular\_arc(double x1, double y1, double x2, double y2)*

**Perl PHP** *circular\_arc(float x1, float y1, float x2, float y2)*

**C** *void PDF\_circular\_arc(PDF \*p, double x1, double y1, double x2, double y2)*

---

Draw a circular arc segment defined by three points.

**x1, y1** The coordinates of an arbitrary point on the circular arc segment.

**x2, y2** The coordinates of the end point of the circular arc segment.

*Details* This function adds a circular arc segment to the current path. The arc segment will start at the current point, pass through  $(x_1, y_1)$ , and end at  $(x_2, y_2)$ . The current point must be set before using this function. The endpoint of the curve 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* *path*

*Params* *currentx, currenty*

---

**C++ Java** *void ellipse(double x, double y, double rx, double ry)*

**Perl PHP** *ellipse(float x, float y, double rx, double ry)*

**C** *void PDF\_ellipse(PDF \*p, double x, double y, double rx, double ry)*

---

Draw an ellipse.

**x, y** The coordinates of the center of the ellipse.

**rx, ry** The *x* and *y* radii of the ellipse.

*Details* This function adds an ellipse to the current path as a complete subpath. The point  $(x + rx, y)$  becomes the new current point.

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

*Params* *currentx, currenty*

---

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

**Perl PHP** `rect(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 line-width (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** `closepath()`

**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*

---

## 7.5 Painting and Clipping

*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** `stroke()`

**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** `closepath_stroke()`

**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** `fill()`

**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** `fill_stroke()`

**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** `closepath_fill_stroke()`

**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** `clip()`

**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** `endpath()`

**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.

---

## 7.6 Path Objects

---

**C++ Java** `int add_path_point(int path, double x, double y, String type, String optlist)`  
**Perl PHP** `int add_path_point(int path, float x, float y, string type, string optlist)`  
**C** `int PDF_add_path_point(PDF *p, int path, double x, double y, const char *type, const char *optlist)`

---

Add a point to a new or existing path object.

**path** A valid path handle returned by another call to `PDF_add_path_point()` or -1 (in PHP: 0) to create a new path.

**x, y** Coordinates of the new current point. If `polar=false` the two numbers designate the cartesian coordinates ( $x, y$ ) of the point. If `polar=true` the two numbers designate the radius  $r$  and angle  $\phi$  (in degrees or radians depending on the option `radians`) of the point.

This point will become the new current point for `type=move, line, curve, circular`.

**type** Specifies the type of the point according to Table 7.6.

Table 7.6 Types of points for `PDF_add_path_point()`

keyword	explanation
<b>circular</b>	Add a circular arc from the current point to the new point with the previously defined control point as third circular arc point which is required. If the new point is identical with the current point a circle with diameter between the current point and the control point will be created.
<b>control</b>	Control point for a Bézier curve or a circular arc.
<b>curve</b>	Add a Bézier curve from the current point to the new point with the previously defined control points. At least one control point must be provided. If only one control point is available, it will be used as the second control point for the curve, and the first control point will be constructed as the reflection of the second control point at the endpoint of the previous Bézier curve.
<b>line</b>	Add a line segment from the current point to the new point.
<b>move</b>	Start a new subpath or (if the appearance changes or a different path operation is used) a new path. Subpaths will be numbered consecutively (1, 2, ...). The first subpath starts at the origin.
<b>origin</b>	New origin for absolute coordinates. If <code>relative=true</code> the coordinates refer to the last origin. Origins can be set arbitrarily often and will not be inserted in the path object. Default: (0, 0)

**optlist** An option list specifying path construction options:

- ▶ Path calculation and naming options for a point according to Table 7.7: `name, polar, radians, relative`
- ▶ Options for assigning attributes to a single subpath (only for `type=move`) according to Table 7.7: `close, fill, round, stroke`
- ▶ Graphics appearance options (only for `type=move`) according to Table 7.2: `dasharray, dashphase, fillcolor, fillrule, flatness, gstate, linecap, linejoin, linewidth, miterlimit, strokecolor`

**Returns** A path handle which can be used until it is deleted with `PDF_delete_path()`.

**Details** A path object serves as a container for vector graphics. The path object can be populated with paths and subpaths incrementally. The generated path can later be used with `PDF_draw_path()` and other functions.

A path object can hold any number of paths: whenever the appearance changes (e.g. a new color) or a different path operation option is used (e.g. stroke vs. fill) a new path is started automatically.

Each path in turn can contain one or more subpaths. A subpath starts at a point with `type=move` and ends before the next point with `type=move`, or before the end of the chain of points.

All subpaths will be closed, filled, stroked, and rounded separately according to the specified options. All paths will be filled separately.

**Scope** any

Table 7.7 Options for `PDF_add_path_point()`

option	explanation
<b>close</b>	(Boolean; only for <code>type=move</code> ) If true, the subpath will be closed with a straight line. Default: see footnote <sup>1</sup>
<b>fill</b>	(Boolean; only for <code>type=move</code> ) If true the subpath will be closed and filled. Default: see footnote <sup>1</sup>
<b>name</b>	(String) Name of the point. Default: <code>p&lt;i&gt;</code> (e.g. <code>p1</code> ) where <code>i</code> is the consecutive number of supplied points.
<b>polar</b>	(Boolean) If true, the <code>(x, y)</code> parameters are polar coordinates specifying radius <code>r</code> and angle <code>phi</code> , otherwise cartesian coordinates specifying <code>x</code> and <code>y</code> values. Default: false
<b>radians</b>	(Boolean) If true, angles for polar coordinates are specified in radians, otherwise in degrees. Default: false
<b>relative</b>	(Boolean) If true, <code>(x, y)</code> are relative to the current point, otherwise to the current origin. Default: see footnote <sup>1</sup>
<b>round</b>	(Float; only for <code>type=move</code> ) Adjacent line vertices in the subpath will be rounded in their joining point by a circular arc with the line segments as its tangents and with the specified radius. If the radius is negative the arc will be grooved so that the corners are circularly grooved. If <code>close=true</code> and no line from the last to the first point was explicitly specified, the first line and the closing line will also be rounded. If <code>round=0</code> no rounding will be done. Default: see footnote <sup>1</sup>
<b>stroke</b>	(Boolean; only for <code>type=move</code> ) If true the subpath will be stroked. Default: see footnote <sup>1</sup>

<sup>1</sup> The default is specified in `PDF_draw_path()`, `PDF_info_path()`, the `textpath` option of `PDF_fit_textline()`, the `wrap` option of `PDF_fit_textflow()`, or the `fitpath` option of `PDF_add_table_cell()`.

**C++ Java** `void draw_path(int path, double x, double y, String optlist)`

**Perl PHP** `draw_path(int path, float x, float y, string optlist)`

**C** `void PDF_draw_path(PDF *p, int path, double x, double y, const char *optlist)`

Draw a path object.

**path** A valid path handle returned by a call to `PDF_add_path_point()` or another function which returns a path handle (e.g. `PDF_info_image()` with the `boundingbox` keyword).

**x, y** Coordinates of the reference point in user coordinates. The reference point is used by various options, and specifies the position of the origin of the path object in the current user system. This implies a translation of the path object.

If the *boxsize* option is specified,  $(x, y)$  is the lower left corner of the fitbox (see Table 6.1) into which the path object will be fit.

**optlist** An option list specifying path drawing options:

- ▶ Fitting options according to Table 6.1:  
*align, attachmentpoint, boxsize, fitmethod, orientate, position, scale*
- ▶ Path operation and subpath selection options according to Table 7.8:  
*clip, close, fill, round, stroke, subpaths*
- ▶ Graphics appearance options according to Table 7.2; these are relevant for the *fill* and *stroke* options:  
*dasharray, dashphase, fillcolor, flatness, gstate, linecap, linejoin, linewidth, miterlimit, strokecolor*
- ▶ Graphics appearance option according to Table 7.2; this is relevant for the *clip* and *fill* options: *fillrule*

**Details** The path(s) will be placed at the reference point  $(x, y)$  and then be stroked, filled, or used as a clipping path according to the specified options. This function does not modify the current graphics state unless the *clip* option is used. The appearance and operation options override the default settings, but they do not override any appearance option which may have been specified for a subpath in *PDF\_add\_path\_point()*.

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

Table 7.8 Path operation options for *PDF\_draw\_path\_point()* for controlling all subpaths in a path object

option	explanation
<i>clip</i>	(Boolean) If true the path will be closed and used as clipping path. Default: false
<i>close</i>	(Boolean) If true, each subpath will be closed with a straight line. Default: the value specified when the path was constructed, or false if no value was specified
<i>fill</i>	(Boolean) If true each path will be filled. Default: the value specified when the path was constructed, or false if no value was specified
<i>round</i>	(Float) For each subpath, adjacent line vertices will be rounded in their joining point by a circular arc with the line segments as its tangents and with the specified radius. If the radius is negative the arc will be grooved so that the corners are circular grooved. If <i>close=true</i> and no line from the last to the first point was explicitly specified, the first line and the closing line will also be rounded. If <i>round=0</i> no rounding will be done. Default: the value specified when the path was constructed, or 0 if no value was specified
<i>stroke</i>	(Boolean) If true the path will be stroked. Default: false
<i>subpaths</i>	(List of integers or single keyword) List with the numbers of subpaths to be drawn. The keyword <i>all</i> specifies all subpaths. Default: <i>all</i>

---

**C++ Java** *double info\_path(int path, String keyword, String optlist)*

**Perl PHP** *float info\_path(int path, string keyword, string optlist)*

**C** *double PDF\_info\_path(PDF \*p, int path, const char \*keyword, const char \*optlist)*

---

Query the results of drawing a path object without actually drawing it.

**path** A valid path handle returned by a call to *PDF\_add\_path\_point()* or another function which returns a path handle (e.g. *PDF\_info\_image()* with the *boundingbox* keyword).

**keyword** A keyword specifying the requested information according to Table 7.9.

Table 7.9 Keywords for `PDF_info_path()`

keyword	explanation
<b>bounding-box</b>	A handle to a path containing the bounding box in user coordinates (relative to the reference point)
<b>numpoints</b>	Number of supplied points. The option subpaths will be ignored.
<b>px, py</b>	The x or y coordinate (in the user coordinate system) of the path point specified in the name option. The option subpaths will be ignored.
<b>width, height</b>	Width and height of the bounding box of the path in user coordinates; linewidth and miterlimit will be ignored.
<b>x1, y1, x2, y2, x3, y3, x4, y4</b>	Position of the i-th rectangle corner (i=1, 2, 3, 4) of the bounding box in user coordinates relative to the reference point

**optlist** An option list specifying path drawing options:

- ▶ All options of `PDF_draw_path()` according to Table 7.8
- ▶ Additional fitting option according to Table 6.1: *refpoint*
- ▶ Additional option according to Table 7.10: *name*

**Returns** The value of some path properties as requested by keyword.

**Details** This function will perform the same calculations as `PDF_add_path_point()`, but will not create any visible output on the page.

**Scope** any

Table 7.10 Options for `PDF_info_path()`

option	explanation
<b>name</b>	Name of a path point for the keys px or py. A default name (e.g. p1) can be used even if an explicit name has been specified in <code>PDF_add_path_point()</code> .

**C++ Java** `void delete_path(int path)`

**Perl PHP** `delete_path(int path)`

**C** `void PDF_delete_path(PDF *p, int path)`

Delete a path object.

**path** A valid path handle returned by a call to `PDF_add_path_point()` or another function which returns a path handle (e.g. `PDF_info_image()` with the *boundingbox* keyword).

**Details** Delete the path object and all associated internal data structures. Note that path objects will not automatically be deleted in `PDF_end_document()`.

**Scope** any



# 8 Color Functions

## 8.1 Setting Color and Color Space

*Cookbook* A full code sample can be found in the *Cookbook* topic `color/starter_color`.

Table 8.1 lists relevant parameter key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 8.1 Color-related keys for `PDF_get/set_parameter()`

key	explanation
<code>preserveold-pantonenames</code>	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
<code>spotcolorlookup</code>	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.

As an alternative to numerical RGB values you can specify RGB colors via their HTML name or hexadecimal values (see »Color«, page 12).

*Cookbook* A full code sample for using RGB color values can be found in the *Cookbook* topic `color/web_colornames`.

- ▶ *cmymk*: 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.
- ▶ *iccbasedgray/rgb/cmymk*: 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 off-set 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* expects device-independent colors in the CIE L\*a\*b\* color space with D50 standard illuminant. Colors are specified by a luminance value in the range 0-100 and two color values *a* and *b* in the range -128 to 127. The *a* component ranges from green to red/magenta (negative values indicate green, positive values indicate magenta),

and the *b* component ranges from blue to yellow (negative values indicate blue, positive values indicate yellow).

- ▶ *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 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 »Color«, page 12, 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 setcolor(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** Specifies the colorspace to be used for the supplied color values, or an RGB color value which is specified by name or hexadecimal values.

First form: one of *gray*, *rgb*, *cmyk*, *spot*, *pattern*, *iccbasedgray*, *iccbasedrgb*, *iccbasedcmyk*, or *lab* to specify the color space.

Second form: an RGB color name (e.g. *pink*) or a hash character followed by six hexadecimal digits (e.g. *#FFCoCB*). See »Color«, page 12, for more details.

**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 with 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 -128 to 127.
- ▶ *pattern*: *c1* specifies a pattern handle returned by *PDF\_begin\_pattern()* or *PDF\_shading\_pattern()*. The current fill or stroke color will be applied when the pattern is used for filling or stroking. The current color space must not be another pattern color space.

*Details* All color values for the *gray*, *rgb*, and *cmymk* 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 *cmymk* 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/cmymk* color spaces are used, a suitable ICC profile must have been set before using the *setcolor:iccprofilegray/rgb/cmymk* parameters (see Table 8.3).

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

PDF/X-3: Using *iccbasedgray/rgb/cmymk* and *lab* color requires an ICC profile in the output intent (a standard name is not sufficient in this case).

PDF/X-3/4/5: *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 output intent is an RGB device.

*colorspace=cmymk* requires that the *defaultcmymk* option in *PDF\_begin\_page\_ext()* has been set unless the output intent is a CMYK device.

PDF/A: *colorspace=gray* requires that the *defaultgray* option in *PDF\_begin\_page\_ext()* has been set unless an output intent (any type) has been specified.

*colorspace=rgb* requires that the *defaultrgb* option in *PDF\_begin\_page\_ext()* has been set unless the output intent is an RGB device.

*colorspace=cmymk* requires that the *defaultcmymk* option in *PDF\_begin\_page\_ext()* has been set unless the output intent is a CMYK device.

*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/cmymk*

---

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

**Perl PHP** *int makespotcolor(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.

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.

**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.

*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*

## 8.2 ICC Profiles

---

**C++ Java** `int load_iccprofile(String profilename, String optlist)`  
**Perl PHP** `int load_iccprofile(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, or a disk-based or virtual file name. If PDF/X-1 or PDF/X-3 is generated and *usage=outputintent*, one of the standard output condition names listed in Table 8.5 (or a name defined with the *Standard-OutputIntent* resource category) can also be used without embedding the corresponding ICC profile. If one of the standard output intents is to be used with PDF/X-4 or PDF/X-5pg, the corresponding ICC profile must be configured as with the *ICCProfile* resource, using the reference name in Table 8.5 as the resource name.

**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 profile handling:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ Profile handling options according to Table 8.2: *description*, *embedprofile*, *metadata*, *urls*, *usage*

Table 8.2 Options for `PDF_load_iccprofile()`

key	explanation and possible values
<b>description</b>	(String; only for <i>usage=outputintent</i> and non-standard output conditions) Human-readable description of the ICC profile which will be used along with the output intent.
<b>embedprofile</b>	(Boolean; only for PDF/X-1 or PDF/X-3 and <i>usage=outputintent</i> ; will be forced to true for PDF/X-4 and PDF/X-5g) Force an embedded ICC profile even if a standard output intent for PDF/X has been supplied as <i>profilename</i> . Default: false
<b>metadata</b>	(Option list; will be ignored for standard output intents and referenced output intents, i.e. <i>usage=outputintent</i> in PDF/X-4p and PDF/X-5pg mode; PDF 1.4) Supply metadata for the profile (see Section 14.2, »XMP Metadata«, page 227)
<b>urls</b>	(List of one or more strings; only for PDF/X-4p and PDF/X-5pg, and required in this case) A list of URLs which indicate where a referenced output intent ICC profile can be obtained. Sender and receiver should arrange reasonable URL entries. The strings can freely be chosen, but must contain valid URL syntax.
<b>usage</b>	(Keyword) Intended use of the ICC profile. Supported keywords (default: <i>iccbased</i> ): <b>iccbased</b> The ICC profile will be used to define an ICC-based color space for text or graphics, or will be applied to an image. Input, display and output device (scanner, monitor, and printer) profiles as well as color space conversion profiles can be used. <b>outputintent</b> The ICC profile will be used to define an output intent for PDF/X or PDF/A. In PDF/X-4p and PDF/X-5pg mode the specified output intent ICC profile will not be embedded, but a reference to an external profile will be created. The profile must be available locally when generating the PDF, and must be available to the PDF consumer when viewing or printing the document. In PDF/X mode only output device (printer) profiles can be used for <i>usage=outputintent</i> .

**Returns** A profile handle which can be used in subsequent calls to `PDF_load_image()` or for setting profile-related parameters. If `errorpolicy=return` the caller must check for a return value of -1 (in PHP: 0) since it signals an error. The returned profile handle can not be re-used across multiple PDF documents. Also, the returned handle can not be applied to an image if `usage=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 `usage=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. Loaded ICC profiles must conform to ICC versions up to 2.x for PDF 1.4, and to ICC versions up to 4.x for PDF 1.5 and above. Profiles can be specified in the gray, RGB, CMYK, or Lab color spaces.

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/X-1 and PDF/X-3: If `usage=outputintent` the named profile is first searched in the internal list of standard output intents and then in the list of user-configured output intents. If the supplied `profilename` was found to be a standard output intent, no ICC profile is required and only the name will be written to the PDF output as 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 (possibly mapped via the `ICCProfile` resource category) will be embedded in the PDF as output intent.

PDF/X-4/5: a CMYK profile which has been loaded with `usage=iccbased` can not be used with `usage=outputintent` in the same document. This requirement is mandated by the PDF/X standard, and applies only to CMYK profiles, but not to grayscale or RGB.

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 8.3 and Table 8.4

Table 8.3 Keys for `PDF_get/set_parameter()` (see Section 2.2, »Parameter and Option Handling«, page 19)

key	explanation
<code>ICCProfile</code>	The corresponding resource file line as it would appear for the respective category in a UPR file.
<code>StandardOutputIntent</code>	Multiple calls add new entries to the internal list (see also <code>resourcefile</code> in Table 2.3). Scope: any

Table 8.4 Keys for `PDF_get/set_value()` (see Section 2.2, »Parameter and Option Handling«, page 19)

key	explanation
<code>icccomponents</code>	Number of color components in the ICC profile referenced by the handle provided in the modifier
<code>setcolor:icc-profilegray</code>	ICC profile which specifies a Gray color space for use with <code>PDF_setcolor()</code> . Scope: document, page, pattern, template, glyph

Table 8.4 Keys for PDF\_get/set\_value() (see Section 2.2, »Parameter and Option Handling«, page 19)

key	explanation
<b>setcolor:icc-profilergb</b>	ICC profile which specifies an RGB color space for use with PDF_setcolor(). Scope: document, page, pattern, template, glyph
<b>setcolor:icc-profilecmk</b>	ICC profile which specifies a CMYK color space for use with PDF_setcolor(). Scope: document, page, pattern, template, glyph

Table 8.5 Standard CMYK output intents for PDF/X-1 and PDF/X-3 (see also www.color.org)

reference name	description
<b>CGATS standards for the US (www.npes.org/standards/tools.html)</b>	
<b>CGATS TR 001</b>	SWOP (Publication) printing in USA: ANSI CGATS.6
<b>CGATS TR 002</b>	SNAP printing in USA
<b>CGATS TR 003</b>	SWOP proofing and printing on U.S. Grade 3 coated publication paper
<b>CGATS TR 005</b>	SWOP proofing and printing on U.S. Grade 5 coated publication paper
<b>CGATS TR 006</b>	GRACoL proofing and printing on U.S. Grade 1 coated paper
<b>FOGRA standards (www.fogra.org)</b>	
<b>FOGRA30</b>	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/m <sup>2</sup> ), screen frequency 60/cm.
<b>FOGRA31</b>	ISO/DIS 12647-2:2003, Continuous forms printing according to ISO 12647-2, positive plates, paper type 2 (matt coated offset, 115 g/m <sup>2</sup> ), screen frequency 60/cm.
<b>FOGRA32</b>	ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, positive plates, paper type 4 (white uncoated offset, 80 g/m <sup>2</sup> ), screen frequency 60/cm.
<b>FOGRA33</b>	ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, positive plates, paper type 2 (matte coated offset, 115 g/m <sup>2</sup> ), screen frequency 54/cm.
<b>FOGRA34</b>	ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, positive plates, paper type 4 (white uncoated offset, 120 g/m <sup>2</sup> ), screen frequency 60/cm.
<b>FOGRA35</b>	ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 2 (matte coated offset, 115 g/m <sup>2</sup> ), screen frequency 54/cm.
<b>FOGRA36</b>	ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 4 (white uncoated offset, 120 g/m <sup>2</sup> ), screen frequency 54/cm.
<b>FOGRA38</b>	ISO/DIS 12647-2:2004, Continuous forms printing according to ISO 12647-2, negative plates, paper type 4 (white uncoated offset, 120 g/m <sup>2</sup> ), screen frequency 60/cm.
<b>FOGRA39</b>	ISO 12647-2:2004 / Amd 1, Offset commercial and specialty printing according to ISO 12647-2, paper type 1 or 2 (gloss or matte coated offset, 115 g/m <sup>2</sup> ), screen frequency 60/cm.
<b>FOGRA40</b>	ISO 12647-2:2004, Offset commercial and specialty printing according to ISO 12647-2, super calendered (SC) paper, 60 g/m <sup>2</sup> , screen frequency 60/cm.
<b>FOGRA41</b>	ISO 12647-2:2004, Offset printing according to ISO 12647-2, paper type MFC (machine finished coated), tone value increase curves B (CMY) and C (K).
<b>FOGRA42</b>	ISO 12647-2:2004, Offset printing according to ISO 12647-2, paper type SNP (Standard Newsprint), tone value increase curves C (CMY) and D (K).
<b>FOGRA43</b>	ISO 12647-2:2004/Amd1, Offset printing according to ISO 12647-2, paper type 1 or 2 (coated art) 115 g/m <sup>2</sup> non-periodic screening, tone value increase curves F (CMYK).

Table 8.5 Standard CMYK output intents for PDF/X-1 and PDF/X-3 (see also [www.color.org](http://www.color.org))

<i>reference name</i>	<i>description</i>
<b>FOGRA44</b>	ISO 12647-2:2004/Amd1, Offset printing according to ISO 12647-2, paper type 4 (uncoated white) 115 g/m <sup>2</sup> non-periodic screening, tone value increase curves F (CMYK).
<b>FOGRA45</b>	ISO 12647-2:2004, Heatset web offset printing according to ISO 12647-2, improved light-weight coated (LWC), 60 l/cm
<b>FOGRA46</b>	ISO 12647-2, Heatset web offset printing according to ISO 12647-2:2004, standard light-weight coated (LWC), 60 l/cm
<b>FOGRA47</b>	ISO 12647-2, Sheetfed offset printing according to ISO 12647-2:2004 / Amd 1, uncoated white (paper type 4), 115 gsm, 60 l/cm
<b>IFRA standards for newsprint (<a href="http://www.ifra.com">www.ifra.com</a>)</b>	
<b>IFRA26</b>	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.
<b>IFRA30</b>	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).
<b>Eurostandard System Brunner (<a href="http://www.systembrunner.com">www.systembrunner.com</a>)</b>	
<b>EUROSB104</b>	Offset printing, according to Eurostandard System Brunner specification, within ISO 12647-2:2004 tolerances, paper type coated/semi-matte, 115 g/m <sup>2</sup> , TVI 15%, screen ruling 60 L/cm, for further information see documentation.
<b>EUROSB204</b>	Offset printing, according to Eurostandard System Brunner specification, within ISO 12647-2:2004 tolerances, LWC woodpulp paper, 80 g/m <sup>2</sup> , TVI 15%, screen ruling 60 L/cm, for further information see documentation.
<b>Japanese standards</b>	
<b>JC200103</b>	Japan Color 2001 Coated: ISO 12647-2:2004, sheet-fed offset printing, positive plates, paper type 3, (coated, 105 g/m <sup>2</sup> ), screen frequency 69/cm.
<b>JC200104</b>	Japan Color 2001 Uncoated: ISO 12647-2:2004, sheet-fed offset printing, positive plates, paper type 4, (uncoated, 105 g/m <sup>2</sup> ), screen frequency 69/cm.
<b>JCN2002</b>	Japan Color 2002 for Newspaper Printing: ISO/DIS 12647-3:2004, coldset offset printing, negative plates, newsprint, screen frequency 39/cm.
<b>JCW2003</b>	Japan Color 2003 for Web Offset: ISO 12647-2:2004, heat-set web offset printing, positive plates, paper type 3, (coated, 70 g/m <sup>2</sup> ), screen frequency 69/cm.

## 8.3 Patterns and Shadings

---

**C++ Java** `int begin_pattern(double width, double height, double xstep, double ystep, int painttype)`  
**Perl PHP** `int begin_pattern(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** This parameter indicates whether the pattern contains color specifications on its own, or is used as a stencil which will be colored with the current fill or stroke color when the pattern is used for filling or stroking:

- ▶ *painttype*=1 must be used if the pattern is colored with one or more calls to `PDF_setcolor()`, or places images or imported PDF pages.
- ▶ *painttype*=2 must be used if the pattern does not contain any color specification. Instead, the current fill or stroke color will be applied when the pattern is used for filling or stroking. Image masks may be used for *painttype*=2. Before using the pattern, `PDF_setcolor()` must be called to set the current color with a color space which is not itself based on another pattern.

Undesired behavior may result if the wrong value of *painttype* is supplied.

**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. Hyper-text 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** `end_pattern()`  
**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 shading_pattern(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 the graphics appearance of the shading pattern according to Table 7.2: *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*

---

**C++ Java** `void shfill(int shading)`

**Perl PHP** `shfill(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*)

---

**C++ Java** `int shading(String shtype, double xo, double yo, double x1, double y1,  
double c1, double c2, double c3, double c4, String optlist)`

**Perl PHP** `int shading(string shtype, float xo, float yo, float x1, float y1,  
float c1, float c2, float c3, float c4, string optlist)`

**C** `int PDF_shading(PDF *p, const char *shtype, double xo, double yo, 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).

**shtype** The type of the shading; must be *axial* for linear shadings or *radial* for circle-like shadings.

---

**$x_0, y_0, x_1, y_1$**  For axial shadings,  $(x_0, y_0)$  and  $(x_1, y_1)$  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.

**$c_1, c_2, c_3, c_4$**  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  $c_1$  will be ignored, and  $c_2$  contains the tint value.

***optlist*** An option list describing aspects of the shading according to Table 8.6. The following options can be used: *antialias*, *boundingbox*, *domain*, *extendo*, *extend1*, *N*, *ro*, *r1*, *startcolor*

**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 8.6 Options for `PDF_shading()`

Option	explanation
<b><i>antialias</i></b>	(Boolean) Specifies whether to activate antialiasing for the shading. Default: false
<b><i>boundingbox</i></b>	(Rectangle) A rectangle defining the shading's bounding box in user coordinates. The bounding box will be applied as a temporary clipping path when the shading is painted (in addition to the current clipping path which may be in effect). This option may be useful to clip the shading without applying <code>PDF_clip()</code> .
<b><i>domain</i></b>	(List of 2 Floats) Two numbers specifying the limiting values of a parametric variable $t$ . The variable is considered to vary linearly between these two values as the color gradient varies between the starting and ending points of the axis. Default: {0 1}
<b><i>extendo</i></b>	(Boolean) Specifies whether to extend the shading beyond the starting point. Default: false
<b><i>extend1</i></b>	(Boolean) Specifies whether to extend the shading beyond the endpoint. Default: false
<b><i>N</i></b>	(Float) Exponent for the color transition function; must be $> 0$ . Default: 1
<b><i>ro</i></b>	(Float; only for radial shadings, and required in this case) Radius of the starting circle
<b><i>r1</i></b>	(Float; only for radial shadings, and required in this case) Radius of the ending circle
<b><i>startcolor</i></b>	(Color) The color of the starting point. This option may be useful to make the function independent of the current color. Default: the current fill color



# 9 Image and Template Functions

Table 9.1 and Table 9.2 list relevant parameter and value key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 9.1 Image-related keys for PDF\_get/set\_parameter()

key	explanation
<b>honoriccprofile</b>	Read ICC color profiles embedded in images, and apply them to the image data. Default: true
<b>renderingintent</b>	The rendering intent for images. Default: Auto. <b>Auto</b> Do not specify any rendering intent in the PDF file, but use the device's default intent instead. Typical use: unknown cases <b>AbsoluteColorimetric</b> 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 <b>RelativeColorimetric</b> 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 <b>Saturation</b> Saturation of the colors will be preserved while the color values may be shifted. Typical use: business graphics <b>Perceptual</b> 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 9.2 Image-related keys for PDF\_get\_value()

key	explanation
<b>imagewidth</b> <b>imageheight</b>	Deprecated, use PDF_info_image() with the imagewidth and imageheight keys.
<b>image:iccprofile</b>	Deprecated, use PDF_info_image() with the iccprofile key.
<b>orientation</b>	Deprecated, use PDF_info_image() with the orientation key.
<b>resx, resy</b>	Deprecated, use PDF_info_image() with the resx and resy keys.

## 9.1 Images

*Cookbook* A full code sample can be found in the *Cookbook* topic `images/starter_image`.

---

```
C++ Java int load_image(String imagetype, String filename, String optlist)
Perl PHP int load_image(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*, *jbig2* (PDF 1.4 and above), *jpeg*, *jpeg2000* (PDF 1.5 and above), *png*, *raw*, or *tiff* offers slight performance advantages. Type *ccitt* is different from a TIFF file which contains CCITT-compressed image data.

**filename** (Name string; will be interpreted according to the global *filenamehandling* option or parameter, see Table 2.2) 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, .jbig2, .jb2, .jpg .jpeg, .jpx, .jpx2, .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 9.3. The following options can be used:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Color-related options: *colorize*, *honoriccprofile*, *iccprofile*, *invert*, *renderingintent*
- ▶ Clipping, masking, and transparency options: *alphachannelname*, *clippingpathname*, *honorclippingpath*, *ignoremask*, *mask*, *masked*
- ▶ Special PDF features for using the image: *georeference*, *iconname*, *template*
- ▶ Options for raw and CCITT images: *bitreverse*, *bpc*, *components*, *height*, *K*, *width*
- ▶ Options for JBIG2 images: *copyglobals*, *imagehandle*
- ▶ Options for processing the image data: *cascadedflate*, *ignoreorientation*, *inline*, *page*, *passthrough*
- ▶ Other options: *interpolate*, *layer*, *metadata*, *OPI-1.3*, *OPI-2.0*

**Returns** An image handle which can be used in subsequent image-related calls. If *errorpolicy=return* the caller must check for a return value of -1 (in PHP: 0) since it 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  $[width \times components \times bpc / 8] \times height$  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 141. 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-1a/3: JBIG2 images are not allowed.

PDF/X-3/4/5: Grayscale images require that the *defaultgray* option in *PDF\_begin\_page\_ext()* must have been set unless the 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 output intent is an RGB device. CMYK images require that the *defaultcmymk* option in *PDF\_begin\_page\_ext()* must have been set unless the output intent is a CMYK device.

PDF/A: Grayscale images require that the *defaultgray* option in *PDF\_begin\_page\_ext()* has been set unless an output intent (any type) has been specified.

RGB images require that the *defaultrgb* option in *PDF\_begin\_page\_ext()* has been set unless the output intent is an RGB device.

CMYK images require that the *defaultcmymk* option in *PDF\_begin\_page\_ext()* has been set unless the output intent is a CMYK device.

*Scope* *document*, *page*, *font*; must always be paired with a matching call to *PDF\_close\_image()*. Loading images in *document* or *font* scope instead of *page* scope offers slight output size advantages.

*Params* See Table 9.1 and Table 9.2

Table 9.3 Options for `PDF_load_image()`

key	explanation
<b>alphachannel-name</b>	(Name string; only for TIFF images; will be ignored if <code>ignoremask=true</code> ) Read the alpha channel with the specified name from the image file and apply it as a soft mask to the image. The named channel must be present in the image file. Default: the first alpha channel in the image
<b>bitreverse</b>	(Boolean; only for <code>imagetype=ccitt</code> ) If true, do a bitwise reversal of all bytes in the compressed data. Default: false
<b>bpc</b>	(Integer; only for <code>imagetype=raw</code> ; required in this case) Number of bits per component; must be 1, 2, 4, or 8. In PDF 1.5, <code>bpc=16</code> is also allowed.
<b>cascadedflate</b>	(Boolean; only for <code>imagetype=jpeg</code> ) If true, an additional layer of Flate compression will be applied to the JPEG-compressed image data. This can reduce output file size in certain cases, e.g. for images with large areas of the same color. Note that for most types of image content this option will not decrease file size, and may even result in larger output. Default: false
<b>clipping-pathname</b>	(String; only for <code>imagetype=tiff</code> and <code>jpeg</code> ; will be ignored if <code>honorclippingpath=false</code> ) 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. The special name <code>Work Path</code> can be used to address a temporary path created in Photoshop. Default: name of the path which is provided as clipping path in the image file
<b>colorize</b>	(Spot color handle; will be ignored if the <code>iccprofile</code> option is provided) Colorize the image with a spot color handle, which must have been retrieved with <code>PDF_makespotcolor()</code> . The image must be a black and white or grayscale image.
<b>components</b>	(Integer; only for <code>imagetype=raw</code> ; required in this case) Number of image components (channels); must be 1, 3, or 4.
<b>copyglobals</b>	(Keyword; only for <code>imagetype=jbig2</code> ) Specify which global segments in a JBIG2 stream will be copied to the PDF. If the JBIG2 stream doesn't contain any global segments this option will not have any effect (default: <code>current</code> ): <ul style="list-style-type: none"> <li><b>all</b> Copy the global segments for all pages in the JBIG2 stream to the PDF. This should be used if more than one page from the same JBIG2 stream will be imported. The <code>imagehandle</code> option should be used if more pages from the same JBIG2 stream will be imported later.</li> <li><b>current</b> Copy only the global segments required for the current page (i.e. the page specified in the page option) in the JBIG2 stream to the PDF. This should be used if no more pages from the same JBIG2 stream will be imported.</li> </ul>
<b>georeference</b>	(Option list; PDF 1.7ext3) Specifies the description of an earth-based coordinate system associated with the image to use for geospatial measuring; see Section 13.2, «Geospatial Features», page 222, for details.
<b>height</b>	(Integer; only for <code>imagetype=raw</code> and <code>ccitt</code> ; required in this case) Image height in pixels.
<b>honor-clippingpath</b>	(Boolean; only for <code>imagetype=tiff</code> and <code>jpeg</code> ) Read the clipping path from the image file if available, and apply it to the image. Default: true
<b>honor-iccprofile</b>	(Boolean; only for <code>imagetype=jpeg</code> , <code>png</code> , and <code>tiff</code> ; will be set to false if the <code>colorize</code> option is provided) Read an embedded ICC profile (if any) and apply it to the image. Default: the value of the <code>honoricc-profile</code> parameter.
<b>iccprofile</b>	(ICC handle; only for <code>imagetype=jpeg</code> , <code>jbig2</code> , <code>png</code> , and <code>tiff</code> ) Handle of an ICC profile which will be applied to the image. Default: an embedded profile if one is present in the image and <code>honoriccprofile=true</code> .
<b>iconname</b>	(Hypertext string; will be ignored if <code>inline=true</code> ; forces <code>template=true</code> ) 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.
<b>ignoremask</b>	(Boolean; must be set to true in PDF/X-1, PDF/X-3, and PDF/A modes for images with an alpha channel) Ignore transparency information and alpha channels in the image. Default: false

Table 9.3 Options for `PDF_load_image()`

key	explanation
<b>ignore-orientation</b>	(Boolean; only for <code>imagetype=tiff</code> ) Ignores any orientation tag in the image. This may be useful for compensating wrong orientation information. Default: false
<b>imagehandle</b>	(Image handle; only for <code>imagetype=jbig2</code> ) Add a reference to an existing global segment attached to another image created from the same JBIG2 stream which must have been loaded earlier with the <code>copy-globals=all</code> option. It is an error to refer to an image which has been created from a different file than the current JBIG2 stream. The specified image handle must not have been closed. Default: no image handle, i.e. a new PDF object will be created with all required global segments for the current page only
<b>inline</b>	(Boolean; only for <code>imagetype=ccitt</code> , <code>jpeg</code> , and <code>raw</code> ) If true, the image will be written directly into the content stream of a page, pattern, template, or glyph description. This option will implicitly call <code>PDF_fit_image()</code> and <code>PDF_close_image()</code> (see PDFlib Tutorial). Using this option is recommended for bitmap glyphs of Type 3 fonts, and should not be used in other situations. If this option is provided, <code>PDF_load_image()</code> can be called in page, pattern, template, glyph scope, and <code>PDF_close_image()</code> must not be called. Default: false
<b>interpolate</b>	(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
<b>invert</b>	(Boolean; not for <code>imagetype=jpeg2000</code> unless <code>mask=true</code> ) 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
<b>K</b>	(Integer; only for <code>imagetype=ccitt</code> ) CCITT parameter for compression scheme selection. Default: 0 <ul style="list-style-type: none"> <li>-1 G4 compression</li> <li>0 One-dimensional G3 compression (G3-1D)</li> <li>1 Mixed one- and two-dimensional compression (G3, 2-D)</li> </ul>
<b>layer</b>	(Layer handle; PDF 1.5) Layer to which the image will belong unless another layer has been activated with <code>PDF_begin_layer()</code> prior to placing the image. Calling <code>PDF_begin_layer()</code> to activate a layer before placing the image overrides the image's layer option. Call <code>PDF_end_layer()</code> before placing the image to make sure that the image's layer option will not be overridden.
<b>mask</b>	(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: <ul style="list-style-type: none"> <li>▶ 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.</li> <li>▶ 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.</li> </ul> This option forces <code>ignoremask=true</code> since an image which is used as mask cannot itself have an internal mask.
<b>masked</b>	(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 <code>PDF_load_image()</code> 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. This option is ignored if the image contains an alpha channel and <code>ignoremask=false</code> . In PDF/A and PDF/X-1/3 mode this option is only allowed with 1-bit masks.
<b>metadata</b>	(Option list; PDF 1.4) Supply metadata for the image (see Section 14.2, »XMP Metadata«, page 227).

Table 9.3 Options for PDF\_load\_image()

key	explanation
<b>OPI-1.3</b>	<p>(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<sup>1</sup>), ALDImageDimensions (list of integers), ALDImageCropRect (rectangle with integers), ALDImagePosition (list of floats)</p> <p>The following entries are optional:</p> <p>ALDImageID (string), ALDObjectComments (string), ALDImageCropFixed (rectangle), ALDImageResolution (list of floats), ALDImageColorType (keyword; one of Process, Spot, Separation; default: Spot), ALDImageColor (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)</p> <p>The suboption <code>normalizefilename</code> controls the handling of file names: if <code>true</code>, file names will be normalized as mandated by the PDF reference. If <code>false</code> 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: <code>false</code></p>
<b>OPI-2.0</b>	<p>(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<sup>1</sup>)</p> <p>The following entries should either both be present or absent:</p> <p>ImageCropRect (rectangle), ImageDimensions (list of floats)</p> <p>The following entries are optional:</p> <p>MainImage (string), TIFFASCIIITag (list of integer/string pairs), ImageOverprint (boolean), ImageInks (the string <code>full_color</code>, the string <code>registration</code>, or a list containing the string <code>monochrome</code> 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)</p> <p>The option <code>normalizefilename</code> is also supported (see OPI-1.3).</p>
<b>page</b>	<p>(Integer; only for <code>imagetype=gif, jbig2, and tiff</code>; 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. The call will fail if the requested page cannot be found in the image file. Default: 1</p>
<b>passthrough</b>	<p>(Boolean; only for <code>imagetype=tiff or jpeg</code>) Controls handling of TIFF and JPEG image data.</p> <p><b>tiff</b> (Default: <code>true</code>) If <code>true</code>, compressed TIFF image data will be directly passed through to the PDF output if possible. Setting this option to <code>false</code> may help in cases where a TIFF image contains damaged or incomplete data.</p> <p><b>jpeg</b> (Default: <code>false</code>) If <code>false</code>, PDFlib will transcode JPEG image data in order to clean up the data for compatibility with Acrobat. If <code>true</code>, 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 <code>true</code> may speed up processing, but certain rare JPEG flavors won't display correctly in Acrobat.</p>
<b>rendering-intent</b>	<p>(Keyword) Rendering intent for the image. See Table 9.1 for a list of possible keywords and their meaning. Default: the value of the global <code>renderingintent</code> parameter</p>
<b>template</b>	<p>(Boolean) If <code>true</code>, generate a PDF Image XObject embedded in a Form XObject (called <code>template</code> 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: <code>false</code>. Scope: document</p>
<b>width</b>	<p>(Integer; only for <code>imagetype=raw and ccitt</code>; required in this case) Image width in pixels</p>

<sup>1</sup> Windows users keep in mind that a sequence of two backslash characters is required in the option list to create a single backslash in the resulting path (see «Common traps and pitfalls», page 9).

---

**C++ Java** `void close_image(int image)`  
**Perl PHP** `close_image(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** `fit_image(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 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 image fitting and processing options. The following options are supported:

- ▶ Fitting options according to Table 6.1:  
*boxsize, blind, dpi, fitmethod, matchbox, orientate, position, rotate, scale, showborder*
- ▶ Options for image processing according to Table 9.4:  
*adjustpage, gstate, ignoreclippingpath, ignoreorientation*

**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` (only if the pattern's *painttype* is 1, or if the image is a mask), `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 9.4 Options for `PDF_fit_image()`, `PDF_fit_pdi_page()`, and `PDF_fill_block()`

key	explanation
<b>adjustpage</b>	<p>(Boolean; only effective in page scope; not allowed if the topdown option has been supplied in <code>PDF_begin_page_ext()</code>) 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 <math>(x, y)</math> with the function parameters <math>x</math> and <math>y</math>. The <code>MediaBox</code> will be adjusted, and all other box entries will be reset to their defaults. With the value <code>o</code> for the position option the following useful cases shall be noted:</p> <p><math>x \geq 0</math> and <math>y \geq 0</math>  The object is surrounded by a white margin. This margin has thickness <math>y</math> in horizontal direction and thickness <math>x</math> in vertical direction.</p> <p><math>x &lt; 0</math> and <math>y &lt; 0</math>  Horizontal and vertical strips will be cropped from the image.</p> <p>Default: false</p>
<b>gstate</b>	<p>(Gstate handle) Handle for a graphics state retrieved with <code>PDF_create_gstate()</code>. The graphics state affects all graphical elements created with this function. Default: no gstate (i.e. current settings will be used)</p>
<b>ignore-clippingpath</b>	<p>(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</p>
<b>ignore-orientation</b>	<p>(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 <code>ignoreorientation</code> option in <code>PDF_load_image()</code></p>

---

**C++ Java** `double info_image(int image, String keyword, String optlist)`

**Perl PHP** `float info_image(int image, string keyword, string optlist)`

**C** `double PDF_info_image(PDF *p, int image, const char *keyword, const char *optlist)`

---

Format an image and query metrics and other image properties.

**image** A valid image or template handle retrieved with one of the `PDF_load_image()` or `PDF_begin_template_ext()` functions.

**keyword** A keyword specifying the requested information according to Table 9.5.

**optlist** An option list specifying options for `PDF_fit_image()`. Options which are not relevant for determining the value of the requested keyword will be ignored.

**Returns** The value of some image property as requested by *keyword*. If the requested property is not available in the image file, the function returns `o`. If an object handle is requested (e.g. *clippingpath*) this function will return a handle to the object, or `-1` (in PHP: `o`) if the object is not available.

**Details** This function performs all calculations required for placing the image according to the supplied options, but will not actually create any output on the page. The image reference point is assumed to be `{o o}`.

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

Table 9.5 Keywords for `PDF_info_image()`

<b>keyword</b>	<b>explanation</b>
<b>boundingbox</b>	Path handle of the image's bounding box
<b>clippingpath</b>	Path handle of the image's clipping path, or -1 (in PHP: o) if no clipping path is present
<b>filename</b>	String index for the name of the image file (including a searchpath directory if applicable), or -1 for templates
<b>fitscalex, fitscaley</b>	Scaling factors which resulted from fitting the image to a box according to the supplied options
<b>height</b>	Image height in user coordinates according to the supplied options
<b>iccprofile</b>	Handle for the ICC profile embedded in the image or -1 (in PHP: o) if no profile is present
<b>imageheight</b>	Images: height in pixels Templates: user-supplied height, or automatically determined height if the reference option has been specified
<b>imagemask</b>	Image handle of the mask associated with the image, or -1 (in PHP: o) if no mask is attached
<b>imagetype</b>	String index for the type (format) of the image: bmp, ccitt, gif, jbig2, jpeg, jpeg2000, png, raw, tiff, or template for templates
<b>imagewidth</b>	Images: width in pixels Templates: user-supplied width, or automatically determined width if the reference option has been specified
<b>mirroringx, mirroringy</b>	Horizontal or vertical mirroring of the image (expressed as 1 or -1) according to the supplied options
<b>orientation</b>	Orientation value of the 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.
<b>resx, resy</b>	Horizontal or vertical resolution of the image. Positive values represent the image resolution in pixels per inch (dpi). The value zero means that the resolution is unknown. Negative values can be used together to determine the aspect ratio of non-square pixels, but don't have any absolute meaning.
<b>strips</b>	Number of image strips (will be different from 1 only for certain multi-strip TIFF images)
<b>targetbox</b>	Deprecated
<b>targetx1, targety1, targetx2, targety2, targetx3, targety3, targetx4, targety4</b>	Deprecated
<b>width</b>	Image width in user coordinates according to the supplied options
<b>x1, y1, x2, y2, x3, y3, x4, y4</b>	Position of the i-th rectangle corner (i=1, 2, 3, 4) of the image bounding box in user coordinates according to the supplied options

## 9.2 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 11, »Block Filling Functions (PPS)«, page 181).

---

**C++ Java** `int begin_template_ext(double width, double height, String optlist)`  
**Perl PHP** `int begin_template_ext(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. The *width* and *height* parameters can be 0. In this case they must be supplied in `PDF_end_template_ext()`. Ultimately both values must be different from 0 unless the *postscript* option is specified.

**optlist** Option list specifying template-related properties.

- ▶ The following options of `PDF_load_image()` can be used (see Table 9.3):  
*iconname, layer, metadata, OPI-1.3, OPI-2.0*
- ▶ The following options of `PDF_begin_page_ext()` can be used (see Table 3.7):  
*topdown, transparencygroup*
- ▶ The *reference* option (see Table 9.6).
- ▶ The following option for including PostScript code can be used (see Table 9.6):  
*postscript*

**Returns** A template handle which can be used in subsequent calls to `PDF_fit_image()` and `PDF_info_image()`, and `PDF_end_template_ext()`, or -1 (in PHP: 0) in case of an error.

**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.

Template size: in the simplest case width and height are supplied in `PDF_begin_template_ext()`. However, if they are not yet known they can also be specified as zero. In this case they must be supplied in the corresponding call to `PDF_end_template_ext()`.

If the *reference* option has been supplied the size will be determined automatically from the size of the target PDF page, and no values must be specified. However, if *width* and *height* are specified nevertheless they will be used, but will automatically be adjusted to the same aspect ratio as the target page.

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

Table 9.6 Options for PDF\_begin\_template\_ext()

key	explanation
<b>postscript</b>	<p>(Option list; not for PDF/A and PDF/X) Create a PostScript XObject instead a PDF Form XObject. This option should only be used in scenarios with tight control over postprocessing of the generated PDF documents; it is not suitable for importing EPS graphics. The PostScript XObject will be written immediately. Although the scope will change to template, no API function calls for producing graphical output are allowed until the corresponding call to PDF_end_template_ext(). If this option is supplied no other options are allowed. Supported suboption:</p> <p><b>filename</b> (Name string; required) Name of a disk-based or virtual file containing PostScript code. The PostScript code should be terminated with a whitespace character. This code will be inserted in the PostScript XObject without any validation. The user is responsible for the PostScript contents.</p>
<b>reference</b>	<p>(Option list; PDF 1.4, but requires Acrobat 9 or above for proper page rendering; not allowed in PDF/X-1/2/3/4 and PDF/A-1 modes; not available in PDFlib source code packages) Specify a reference to a page in an external PDF (the »target« document). The page or template will be used as a proxy for this reference. Depending on viewer configuration and availability of the target PDF either the internal proxy or the external target will be displayed and printed. See Table 9.7 for available suboptions.</p> <p>The target PDF must be available locally and must contain the page addressed with the pagelabel or pagenumber suboption. The target must not require a user or master password. The size of the reference page will be determined according to the pdiusebox suboption of the reference option.</p> <p>PDF_begin_template_ext(): If width and height have been supplied with the value 0 the template size can be retrieved with the imagewidth/imageheight keywords of PDF_info_image(). If width and height have been supplied with values different from 0, the following suboptions can also be used (see Table 6.1): fitmethod, position.</p> <p>In PDF/X-5g or PDF/X-5pg mode the target must conform to one of the following standards: PDF/X-1a:2003, PDF/X-3:2003, PDF/X-4, PDF/X-4p, PDF/X-5g, or PDF/X-5pg, and must have been prepared for the same output intent.</p>

Table 9.7 Suboptions for the reference option in PDF\_begin\_template\_ext() and PDF\_open\_pdi\_page()

key	explanation
<b>filename</b>	<p>(Name string; required) Name of the file containing the target PDF. This name will be stored in the PDF and used by the viewer. It will also be used to locate the target PDF locally (i.e. the PDF must exist) unless the target option has been supplied. It is recommended to use plain base names without any directories.</p>
<b>hypertext-encoding</b>	<p>(Keyword) Specifies the encoding for the pagelabel option. An empty string is equivalent to unicode. Default: value of the global hypertextencoding parameter</p>
<b>pagelabel</b>	<p>(Hypertext string; must not be combined with pagenumber) Page label of the page to be referenced</p>
<b>pagenumber</b>	<p>(Integer) The number of the page to be referenced. The first page has page number 1. Default: 1 (this may be overwritten by pagelabel, however).</p>
<b>pdiusebox</b>	<p>(Keyword; will be forced to media in PDF/X-5g and PDF/X-5pg mode) Specifies which box dimensions will be used for determining the size of the target page. Default: media in PDF/X-5g and PDF/X-5pg mode, else crop.</p> <p><b>media</b> Use the MediaBox (which is always present)</p> <p><b>crop</b> Use the CropBox if present, else the MediaBox</p> <p><b>bleed</b> Use the BleedBox if present, else the CropBox</p> <p><b>trim</b> Use the TrimBox if present, else the CropBox</p> <p><b>art</b> Use the ArtBox if present, else the CropBox</p>

Table 9.7 Suboptions for the reference option in `PDF_begin_template_ext()` and `PDF_open_pdi_page()`

key	explanation
<b>strongref</b>	(Boolean; will be forced to <code>true</code> in PDF/X-5g and PDF/X-5pg mode) If <code>true</code> , PDFlib will use the target's ID entry to create a strong reference to the target. The reference will break (i.e. the viewer will use the proxy) if the target is replaced with another document. If the flexibility of swapping targets is desired, this option must be set to <code>false</code> , and the local target and the target which is ultimately used for rendering the document must have identical page boxes and rotation entries. Default: <code>true</code>
<b>target</b>	(PDF document handle) A handle to the target document retrieved with <code>PDF_open_pdi_document()</code> . The target PDF must have been opened with the <code>repair=none</code> option and without the password option. Supplying a document handle in addition to the filename may be useful in two situations: <ul style="list-style-type: none"> <li>▶ If many generated documents reference the same target PDF, the target must be opened only once and the results can be cached internally.</li> <li>▶ The filename of the local target is different from the target filename to be stored in the PDF.</li> </ul>

---

**C++ Java** `void end_template_ext(double width, double height)`

**Perl PHP** `end_template_ext(float width, float height)`

**C** `void PDF_end_template_ext(PDF *p, double width, double height)`

---

Finish a template definition.

**width, height** The dimensions of the template's bounding box in points. If *width* or *height* is 0, the value supplied in `PDF_begin_template_ext()` will be used. Otherwise the value supplied in `PDF_begin_template_ext()` will be overwritten. However, if the reference option has been specified in the corresponding call to `PDF_begin_template_ext()`, the values supplied to `PDF_end_template_ext()` will be ignored.

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

---

**C++ Java** `void end_template()`

**Perl PHP** `end_template()`

**C** `void PDF_end_template(PDF *p)`

---

Deprecated, use `PDF_end_template_ext()`.

---

**C++ Java** `int begin_template(double width, double height)`

**Perl PHP** `int begin_template(float width, float height)`

**C** `int PDF_begin_template(PDF *p, double width, double height)`

---

Deprecated, use `PDF_begin_template_ext()`.

## 9.3 Thumbnails

---

**C++ Java** `void add_thumbnail(int image)`  
**Perl PHP** `add_thumbnail(int image)`  
**C** `void PDF_add_thumbnail(PDF *p, int image)`

---

Add an 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 and Adobe Reader generate thumbnails on the fly, 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.



# 10 PDF Import (PDI) and pCOS Functions

*Note* All functions described in this chapter require the PDF import library (PDI) which is included in PDFlib+PDI and PDFlib Personalization Server (PPS), but not in the base PDFlib product. Please visit our Web site for more information on obtaining PDI.

## 10.1 Document Functions

*Cookbook* A full code sample can be found in the *Cookbook* topic `pdf_import/starter_pdfmerge`.

Table 10.1 lists relevant parameter key names for this section (see Section 2.2, »Parameter and Option Handling«, page 19).

Table 10.1 PDI-related keys for `PDF_get/set_parameter()`

key	explanation
<code>pd<sup>1</sup></code>	Returns the string <code>true</code> if PDI has been included when building the underlying library. This is true for all combined PDFlib, PDFlib+PDI, and PPS binaries distributed by PDFlib GmbH, regardless of the license key. Otherwise it returns <code>false</code> . Scope: any, null <sup>2</sup>

1. Only for `PDF_get_parameter()`

2. May be called with a `PDF *` argument of `NULL` or `o`

---

**C++ Java** `int open_pdi_document(String filename, String optlist)`

**Perl PHP** `int open_pdi_document(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; will be interpreted according to the global `filenamehandling` option or parameter, see Table 2.2) The name of the PDF file.

**optlist** An option list specifying PDF open options:

- ▶ General option: `errorpolicy` (see Section 2.5, »Exception Handling«, page 27)
- ▶ PDF document options according to Table 10.2:  
`infomode`, `inmemory`, `password`, `repair`, `requiredmode`

**len** (C language binding only) Length of `filename` (in bytes) for UTF-16 strings. If `len = o` 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: `o`) 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()`.

The error behavior can be changed with the `errorpolicy` parameter or option.

**Details** By default, the document will be rejected if at least one of the following conditions is true:

- ▶ The document is damaged and couldn't be repaired (or `repair=none` was specified).

- ▶ 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 any*: in *object* scope a PDI document handle can only be used in the *PDF\_pcos\_get\_\**() functions.

Table 10.2 Options for *PDF\_open\_pdi\_document()*

key	explanation
<b>infomode</b>	(Boolean) If <i>true</i> , the document will be opened such that information can be queried with the <i>pCOS</i> interface, but the pages can not be imported into the current output document. In particular, the following kinds of documents can be opened when <i>infomode=true</i> : <ul style="list-style-type: none"> <li>▶ PDFs which are not compatible to the current PDF/X or PDF/A conformance level</li> <li>▶ Encrypted PDFs where the password is not known (exception: PDF 1.6 documents created with the Distiller setting »Object Level Compression: Maximum«)</li> <li>▶ Tagged PDF when the <i>tagged</i> option in <i>PDF_begin_document()</i> is <i>true</i></li> </ul> Default: <i>false</i> if <i>requiredmode=full</i> , otherwise <i>true</i>
<b>inmemory</b>	(Boolean) If <i>true</i> , 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 z/OS) at the expense of memory usage. If <i>false</i> , individual parts of the document will be read from disk as needed. Default: <i>false</i>
<b>password</b>	(String with a maximum length of 32 characters) Master password required to open a protected PDF document for import. If <i>infomode=true</i> 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.
<b>repair</b>	(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. Supported keywords (default: <i>auto</i> ): <ul style="list-style-type: none"> <li><b>auto</b> Repair the document only if problems are detected while opening the PDF.</li> <li><b>force</b> Unconditionally try to repair the document, regardless of whether or not it has problems.</li> <li><b>none</b> No attempt will be made at repairing the document. If there are problems in the PDF the function call will fail.</li> </ul>
<b>requiredmode</b>	(Keyword) The minimum <i>pcos</i> mode ( <i>minimum/restricted/full</i> ) which is acceptable when opening the document. The call will fail if the resulting <i>pcos</i> mode would be lower than the required mode. If the call succeeds it is guaranteed that the resulting <i>pcos</i> mode is at least the one specified in this option. However, it may be higher; e.g. <i>requiredmode=minimum</i> for an unencrypted document will result in <i>full</i> mode. Default: <i>full</i>

---

**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 10.2. The following options can be used:

*infomode, inmemory, password, 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** `close_pdi_document(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 any*

## 10.2 Page Functions

---

**C++ Java** *int open\_pdi\_page(int doc, int pagenumber, String optlist)*

**Perl PHP** *int open\_pdi\_page(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-specific options:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Page options according to Table 10.3:  
*boxexpand*, *clippingarea*, *cloneboxes*, *forcebox*, *iconname*, *layer*, *metadata*, *pdiusebox*, *reference*
- ▶ The following option of *PDF\_begin\_page\_ext()* (see Table 3.7): *transparencygroup*

**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. If the function call fails you can request the reason of the failure with *PDF\_get\_errmsg()*. The returned handle 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()*.

The error behavior can be changed with the *errorpolicy* parameter or option.

**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.

This function fails if the document uses a PDF version which is incompatible to the current PDF document. For PDF versions up to PDF 1.6 all versions up to and including the same version are compatible. For PDF 1.7 and 1.7 extension level 3 all versions up to PDF 1.7, PDF 1.7 extension level 3 (Acrobat 9), and PDF 1.7 extension level 8 (Acrobat X) are compatible (note that Acrobat X encryption is not yet supported). However, in PDF/A mode the input PDF version number is ignored since PDF version headers must be ignored in PDF/A.

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()*.

If the imported page contains referenced XObjects, *PDF\_open\_pdi\_page()* will copy both proxy and reference to the target.

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.

PDF/A and PDF/X: this call may fail if the document containing the imported page uses an output intent which is incompatible to the generated document.

PDF/X-4/5: the imported page is rejected if it uses a CMYK ICC profile which is identical to the generated document's output intent profile.

**Scope** *document, page*

Table 10.3 Options for PDF\_open\_pdi\_page()

key	explanation
<b>boxexpand</b>	(Float or list with four floats) Expand the page box selected via the pdiusebox option on all four sides by the same amount (if one value is provided) or on the left/bottom/right/top sides individually (if four values are provided). Negative values are allowed to reduce the page size. This option may be used to place content which is located outside of all page boxes of the imported page, or to add margins. Default: 0
<b>clippingarea</b>	(Keyword) Specify which of the page boxes of the imported page will be used for clipping. Content outside the specified area will not be visible after placing the imported page on a new page. Supported keywords (default: pdiusebox): <ul style="list-style-type: none"> <li><b>art</b> Use the ArtBox if present, else the CropBox</li> <li><b>bleed</b> Use the BleedBox if present, else the CropBox</li> <li><b>crop</b> Use the CropBox if present, else the MediaBox</li> <li><b>media</b> Use the MediaBox (which is always present)</li> <li><b>pdiusebox</b> Use the box specified in the pdiusebox option</li> <li><b>trim</b> Use the TrimBox if present, else the CropBox</li> </ul>
<b>cloneboxes</b>	(Boolean; not allowed if boxexpand, forcebox, or pdiusebox is supplied; must match the cloneboxes option in PDF_fit_pdi_page()) If true, the page will be prepared for box cloning with the cloneboxes option of PDF_fit_pdi_page(). Default: false
<b>forcebox</b>	(Rectangle) Force the page box to the specified values. This option overrides the pdiusebox and boxexpand options. It may be used to place content which is located outside of all page boxes of the imported page. The values must be chosen carefully if the imported page contains a /Rotate key. The boxexpand option is preferable since it works regardless of any /Rotate key. Default: the box selected with the pdiusebox option
<b>iconname</b>	(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.
<b>layer</b>	(Layer handle; PDF 1.5) Layer to which the page will belong unless another layer has been activated with PDF_begin_layer() prior to placing the page. Calling PDF_begin_layer() to activate a layer before placing the page overrides the page's layer option. Call PDF_end_layer() before placing the page to make sure that the page's layer option will not be overridden.
<b>metadata</b>	(Option list; PDF 1.4) Supply metadata for the imported page (see Section 14.2, »XMP Metadata«, page 227)
<b>pdiusebox</b>	(Keyword; not allowed if cloneboxes is supplied) Specifies which box dimensions will be used for determining an imported page's size. The box size will be used for scaling operations in PDF_fit_pdi_page(). This box will also determine the visible contents of the page unless modified with the clippingarea option. Default: crop. <ul style="list-style-type: none"> <li><b>art</b> Use the ArtBox if present, else the CropBox</li> <li><b>bleed</b> Use the BleedBox if present, else the CropBox</li> <li><b>crop</b> Use the CropBox if present, else the MediaBox</li> <li><b>media</b> Use the MediaBox (which is always present)</li> <li><b>trim</b> Use the TrimBox if present, else the CropBox</li> </ul>
<b>reference</b>	(Option list; PDF 1.4, but requires Acrobat 9 or above for proper page rendering) Define the page as a proxy which carries a reference to a page in an external PDF document (the target page). The page opened with the current call will be used as a proxy for the referenced target page (see Table 9.6 for details). The proxy page and the target page must have compatible page geometry, i.e. the page boxes selected with the pdiusebox option must be identical to make sure that both pages will be placed at the same location on the page.

---

**C++ Java** `void close_pdi_page(int page)`

**Perl PHP** `close_pdi_page(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** `fit_pdi_page(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 output 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 page options:

- ▶ Fitting options according to Table 6.1:  
`blind, boxsize, fitmethod, matchbox, orientate, position, rotate, scale, showborder`
- ▶ Options for page processing according to Table 9.4: `adjustpage, gstate`
- ▶ The `cloneboxes` option according to Table 10.4.

**Details** This function is similar to `PDF_fit_image()`, but operates on imported PDF pages instead. The following option for `PDF_begin/end_page_ext()` is recommended to improve the output quality if an imported page contains `ExtGState` objects:  
`transparencygroup={colorspace=DeviceRGB}`.

**Scope** `page, pattern, template, glyph`

---

**C++ Java** `double info_pdi_page(int page, String keyword, String optlist)`

**Perl PHP** `float info_pdi_page(int page, string keyword, string optlist)`

**C** `double PDF_info_pdi_page(PDF *p, int page, const char *keyword, const char *optlist)`

---

Perform formatting calculations for a PDI page and query the resulting metrics.

**page** A valid page handle retrieved with `PDF_open_pdi_page()`.

**keyword** A keyword specifying the requested information according to Table 10.5.

**optlist** An option list specifying scaling and placement details:

- ▶ General option: `errorpolicy` (see Section 2.5, »Exception Handling«, page 27)

Table 10.4 Additional option for `PDF_fit_pdi_page()`

key	explanation
<b>cloneboxes</b>	<p>(Boolean; not allowed if the <code>topdown</code> option has been supplied in <code>PDF_begin_page_ext()</code>; must match the <code>cloneboxes</code> option in <code>PDF_open_pdi_page()</code>; only in page scope).</p> <p>Setting this option to true has the following consequences (Default: false):</p> <ul style="list-style-type: none"> <li>▶ All of the <code>Rotate</code>, <code>MediaBox</code>, <code>TrimBox</code>, <code>ArtBox</code>, <code>BleedBox</code> and <code>CropBox</code> entries which are present in the imported page will be copied to the current output page.</li> <li>▶ The page contents will be placed such that the input page is duplicated; the user cannot change position or size of the placed page. The parameters <code>x</code>, <code>y</code> and the following options will therefore be ignored: <code>adjustpage</code>, <code>boxsize</code>, <code>fitmethod</code>, <code>orientate</code>, <code>position</code>, <code>rotate</code>, <code>scale</code>. Duplication of the input page is only possible if the default coordinate system is active when calling <code>PDF_fit_pdi_page()</code>.</li> <li>▶ Page boxes created by the <code>cloneboxes</code> option overrides the <code>artbox</code>, <code>bleedbox</code>, <code>cropbox</code>, <code>trimbox</code>, <code>mediabox</code>, and <code>rotate</code> options as well as the width and height parameters of <code>PDF_begin_page_ext()</code>.</li> </ul> <p>▶ Fitting options according to Table 6.1 (if the PDF page has been opened with the <code>cloneboxes</code> option of <code>PDF_open_pdi_page()</code> these options will be ignored): <code>boxsize</code>, <code>fitmethod</code>, <code>matchbox</code>, <code>orientate</code>, <code>position</code>, <code>rotate</code>, <code>scale</code></p> <p>▶ Options for page processing according to Table 9.4 don't make sense; they will be ignored to facilitate unified option lists for <code>PDF_fit_pdi_page()</code> and <code>PDF_info_pdi_page()</code>: <code>adjustpage</code>, <code>gstate</code></p>

**Returns** The value of some page metrics as requested by *keyword*. If `errorpolicy=return` this function will return 0 (in PHP: 0) in case of an error. If `errorpolicy=exception` this function will throw an exception in case of an error.

**Details** This function performs all calculations required for placing the imported page according to the supplied options, but will not actually create any output on the page. The reference point for placing the page is assumed to be {0 0}. If the `cloneboxes` option of `PDF_open_pdi_page()` has been supplied, the page will be placed on the same location (relative to the page boxes) as in the original page.

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

Table 10.5 Keywords for `PDF_info_pdi_page()`

keyword	explanation
<b>boundingbox</b>	Path handle for the page's bounding box
<b>fitscalex, fitscaley</b>	Scaling factors which resulted from fitting the page to a box according to the supplied options
<b>height</b>	Page height in user coordinates according to the supplied options and the options used in <code>PDF_open_pdi_page()</code>
<b>mirroringx, mirroringy</b>	Horizontal or vertical mirroring of the page (expressed as 1 or -1) according to the supplied options
<b>pageheight</b>	Original page height in points
<b>pagewidth</b>	Original page width in points
<b>rotate</b>	<p>If <code>cloneboxes=true</code>: the rotation angle of the imported page in degrees, i.e. the value of the page's <code>Rotate</code> key. Possible values are 0, 90, 180, and 270).</p> <p>If <code>cloneboxes=false</code>: always 0</p>

Table 10.5 Keywords for PDF\_info\_pdi\_page()

<b>keyword</b>	<b>explanation</b>
<b>width</b>	Page width in user coordinates according to the supplied options and the options used in PDF_open_pdi_page()
<b>x1, y1, x2, y2, x3, y3, x4, y4</b>	Position of the <i>i</i> -th rectangle corner ( <i>i</i> =1, 2, 3, 4) of the page bounding box in user coordinates according to the supplied options. If cloneboxes=true the visible box will be used (i.e. the CropBox if present, else the MediaBox).

## 10.3 Other PDI Processing

---

**C++ Java** *int process\_pdi(int doc, int page, String optlist)*  
**Perl PHP** *int process\_pdi(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 10.6), *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 (in PHP: 0).

**optlist** An option list specifying PDI processing options:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ PDI processing options according to Table 10.6. The following option can be used:  
*action*

**Returns** The value 1 if the function succeeded, or an error code of -1 (in PHP: 0) if the function call failed. If *errorpolicy=exception* this function will throw an exception in case of an error.

**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 10.6 Options for *PDF\_process\_pdi()*

key	explanation
<b>action</b>	(Keyword; required; this option does not require a page handle) Specifies the kind of PDF processing: <b>copyoutputintent</b> (Doesn't do anything if the output document neither conforms to PDF/X nor PDF/A.) 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. If the input and output documents conform to PDF/X-4p or PDF/X-5pg the reference to the external output intent ICC profile will be copied. The option <i>action=copyoutputintent</i> is not allowed if the input conforms to PDF/X-4p or PDF/X-5pg, but not the output.

## 10.4 pCOS Functions

All pCOS functions work with paths designating the target object in the PDF document. pCOS paths are discussed in detail in the *pCOS Path Reference*.

*Cookbook* A full code sample for using pCOS within PDFlib+PDI or PPS can be found in the Cookbook topic `pdf_import/starter_pcos`. A large number of pCOS programming samples is available in the *pCOS Cookbook*.

*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 pcos_get_number(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** `string pcos_get_string(int doc, string path)`  
**Perl PHP** `string pcos_get_string(long doc, string path)`  
**C** `const char *PDF_pcos_get_string(PDF *p, 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 *pages[...]/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 language binding: The string will be returned in UTF-8 format (on zSeries and i5/iSeries: EBCDIC-UTF-8) without BOM. The returned strings will be stored in a ring buffer with up to 10 entries. If more than 10 strings are queried, buffers will be reused, which means that clients must copy the strings if they want to access more than 10 strings in parallel. For example, up to 10 calls to this function can be used as parameters for a *printf()* statement since the return strings are guaranteed to be independent if no more than 10 strings are used at the same time.

C++ language binding: The string will be returned as *wstring* in the default *wstring* configuration of the C++ wrapper. In *string* compatibility mode on zSeries and i5/iSeries the result will be returned in EBCDIC-UTF-8 without BOM.

Java, .NET, and Python: the result will be provided as Unicode string. If no more text is available a null object will be returned.

Perl and PHP language bindings: the result will be provided as UTF-8 string. If no more text is available a null object will be returned.

RPG language binding: the result will be provided as EBCDIC-UTF-8 string.

---

**C++ Java** `const unsigned char *pcos_get_stream(int doc, int *length, string optlist, string path)`

**Perl PHP** `string pcos_get_stream(long doc, string optlist, string path)`

**C** `const unsigned char *PDF_pcos_get_stream(PDF *p, 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** An option list specifying stream retrieval options according to Table 10.7.

**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: NULL) if the stream or string 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) unless *keepfilter=true*. 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 language binding: The returned data buffer can be used until the next call to this function.

Python: the result will be returned as 8-bit string (Python 3: *bytes*).

**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.*

Table 10.7 Options for *PDF\_pcos\_get\_stream()*

option	description
<b>convert</b>	(Keyword; will be ignored for streams which are compressed with unsupported filters) Controls whether or not the string or stream contents will be converted (default: none): <b>none</b> Treat the contents as binary data without any conversion. <b>unicode</b> Treat the contents as textual data (i.e. exactly as in <i>PDF_pcos_get_string()</i> ), and normalize it to Unicode. In non-Unicode-aware language bindings this means the data will be converted to UTF-8 format without BOM. This option is required for the data type »text stream« in PDF which is rarely used (e.g. it can be used for JavaScript, although the majority of JavaScripts is contained in string objects, not stream objects).
<b>keepfilter</b>	(Boolean; recommended only for image data streams; will be ignored for streams which are compressed with unsupported filters) If <i>true</i> , the stream data will be compressed with the filter which is specified in the image's <i>filterinfo</i> pseudo object. If <i>false</i> , the stream data will be uncompressed. Default: <i>true</i> for all unsupported filters, <i>false</i> otherwise



# 11 Block Filling Functions (PPS)

The PDFlib Personalization Server (PPS) offers dedicated functions for processing variable Blocks of type *Text*, *Image*, and *PDF*. These PDFlib 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 PDFlib Blocks in PDF templates.

*Cookbook* A full code sample can be found in the Cookbook topic `blocks/starter_block`.

## 11.1 Rectangle Options for Block Filling Functions

Table 11.1 lists rectangle options for `PDF_fill_textblock()`, `PDF_image_block()`, and `PDF_fill_pdfblock()`. Options which are specific for a particular Block type (i.e. text, image, or PDF Blocks) are listed in the next sections. Almost all Block properties can be overridden with options with the same name, except for the following properties which can not be overridden with options:

Name, Description, Subtype, Type  
`defaulttext`, `defaultimage`, `defaultpdf`, `defaultpdfpage`

Table 11.1 Rectangle options for the `PDF_fill_*block()` functions

key	explanation
<b>Rect</b>	(Rectangle) The coordinates of the Block in the coordinate system of the Block PDF. The Block rectangle can be specified with the <code>repoint</code> and <code>boxsize</code> options (in user coordinates).
<b>Status</b>	(Keyword) Describes how the Block will be processed (default: active): <b>active</b> The Block will be fully processed according to its properties. <b>ignore</b> The Block will be ignored. <b>ignoredefault</b> Like <b>active</b> , except that the <code>defaulttext/image/pdf</code> properties will be ignored, i.e. the Block remains empty if no default contents have been provided. This may be useful to make sure that the Block's default contents will not be used for filling Blocks on the server side although the Block may contain default contents for the Preview in the Block Plugin. It can also be used to disable the default contents for previewing a Block without removing it from the Block properties. <b>static</b> No variable contents will be placed; instead, the Block's default text, image, or PDF contents will be used if available.
<b>background-color</b>	(Color) Fill color for the Block; this color will be applied before filling the Block. This may be useful to cover existing page contents. Default: none
<b>bordercolor</b>	(Color) Border color for the Block; this color will be applied before filling the Block. Default: none
<b>linewidth</b>	(Float; must be greater than 0) Stroke width of the line used to draw the Block rectangle; only used if <code>bordercolor</code> is set. Default: 1

## 11.2 Textline and Textflow Blocks

---

```
C++ Java int fill_textblock(int page, String blockname, String text, String optlist)
Perl PHP int fill_textblock(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 Textline or Textflow Block with variable data according to its properties.

**page** A valid PDF page handle for a page containing PDFlib Blocks. The input PDF page with Blocks must have been placed on the page earlier, either directly with *PDF\_fit\_pdi\_page()*, indirectly in a table cell with *PDF\_fit\_table()*, or as contents of a PDF Block with *PDF\_fill\_pdfblock()*.

**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 *textflowhandle* 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 text Block filling options. The following options are supported:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ Rectangle options for Block filling functions according to Table 11.1:  
*Rect, Status, backgroundcolor, bordercolor, linewidth*
- ▶ Fitting options (see Section 6.1, »Object Fitting«, page 109)
- ▶ Textline Blocks, i.e. the *textflow* property or option is *false*:  
all Textline options (see Section 5.2, »Single-Line Text with Textlines«, page 79);
- ▶ Textflow Blocks, i.e. the *textflow* property or option is *true*:  
all options for *PDF\_add/create\_textflow()* (see Section 5.3, »Multi-Line Text with Textflows«, page 83) and all options for *PDF\_fit\_textflow()* (see Table 5.14)
- ▶ Text Block options according to Table 11.2: *textflow, textflowhandle*

**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 (unless *Status=ignoredefault*), and silently return otherwise. This may be useful to take advantage of other Block properties, such as fill or stroke color.

Font selection: If neither the *font* option is supplied nor implicit font loading based on options is used, the font will be implicitly loaded based on the Block properties. Since the encoding for the font can only be specified as an option, but not as a Block property it will be set as follows by default:

- ▶ *builtin* if the font is a symbolic font and *charref=false* and (only relevant for non-Unicode aware languages) *textformat=auto* or *bytes*.
- ▶ *unicode* otherwise.

It is recommended to avoid the *encoding*, *charref* and *textformat* options if *defaulttext* is to be used.

Special care should be taken regarding the *embedding* option: if the font is implicitly loaded based on Block properties it will not automatically be embedded. If font embedding is desired the *embedding* option must be specified.

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

Table 11.2 Additional options for *PDF\_fill\_textblock()*

key	explanation
<b>textflow</b>	(Boolean) Control single- or multiline processing. This property can be used to switch between Textline and Textflow Blocks: <b>false</b> Text can span a single line and will be processed with <i>PDF_fit_textline()</i> . <b>true</b> Text can span multiple lines and will be processed with <i>PDF_fit_textflow()</i> . The default depends on the Block type: true for Textflow Blocks, and false for Textline Blocks
<b>textflow-handle</b>	(Textflow handle; only for <i>PDF_fill_textblock()</i> with <i>textflow=true</i> ) This option can be used for Textflow Block chaining. For the first Block in a chain of Blocks 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 for other Blocks in the chain. This option will change the default of <i>fitmethod</i> to <i>clip</i> . Note that all properties in the Text Preparation, Text Formatting and Appearance property groups of the Block will be ignored if <i>textflow-handle</i> is supplied since the corresponding values used for creating the Textflow will be applied.

## 11.3 Image Blocks

---

**C++ Java** *int fill\_imageblock(int page, String blockname, int image, String optlist)*  
**Perl PHP** *int fill\_imageblock(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 PDFlib Blocks. The input PDF page with Blocks must have been placed on the page earlier, either directly with *PDF\_fit\_pdi\_page()*, indirectly in a table cell with *PDF\_fit\_table()*, or as contents of a PDF Block with *PDF\_fill\_pdfblock()*.

**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 image Block filling options. The following options are supported:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ Rectangle options for Block filling functions according to Table 11.1:  
*Rect, Status, backgroundcolor, bordercolor, linewidth*
- ▶ Fitting options (see Section 6.1, »Object Fitting«, page 109)
- ▶ Options for image processing according to Table 9.4.

**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 (unless *Status=ignoredefault*), 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*

## 11.4 PDF Blocks

---

**C++ Java** *int fill\_pdfblock(int page, String blockname, int contents, String optlist)*  
**Perl PHP** *int fill\_pdfblock(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 PDFlib Blocks. The input PDF page with Blocks must have been placed on the page earlier, either directly with *PDF\_fit\_pdi\_page()*, indirectly in a table cell with *PDF\_fit\_table()*, or as contents of a PDF Block with *PDF\_fill\_pdfblock()*.

**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 PDF Block filling options. The following options are supported:

- ▶ General option: *errorpolicy* (see Section 2.5, »Exception Handling«, page 27)
- ▶ Rectangle options for Block filling functions according to Table 11.1:  
*Rect, Status, backgroundcolor, bordercolor, linewidth*
- ▶ Fitting options (see Section 6.1, »Object Fitting«, page 109)
- ▶ Options for page processing according to Table 9.4.

**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 (unless *Status=ignoredefault*), 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*



# 12 Interactive Features

## 12.1 Parameters for Interactive Elements

Table 12.1 lists relevant parameter key names for interactive elements (see Section 2.2, »Parameter and Option Handling«, page 19). These parameters are not available in Unicode-aware language bindings.

Table 12.1 String-related keys for `PDF_get/set_parameter()`

key	explanation
<b>hypertextencoding</b>	(Also available as option) Encoding for options and parameters of type hypertext. An empty string is equivalent to unicode. Default: auto. Scope: any
<b>hypertextformat</b>	Format for options and parameters of type hypertext. Possible values are bytes, utf8, utf16, utf16le, utf16be, and auto. Default: auto. Scope: any
<b>usehypertextencoding</b>	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
<b>usercoordinates</b>	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

## 12.2 Actions

---

```
C++ Java int create_action(String type, String optlist)
Perl PHP int create_action(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 action type according to Table 12.2.

Table 12.2 Action types

type	notes; options specific for this type (in addition to general options)
<b>GoTo</b>	Go to a destination in the current document: destination, destname
<b>GoTo3DView</b>	(PDF 1.6) Set the current view of a 3D animation: 3Dview, target
<b>GoToE</b>	(PDF 1.6) Go to a destination in an embedded document: targetpath
<b>GoToR</b>	Go to a destination in another (remote) document: destination, destname, filename, newwindow
<b>Hide</b>	(Not for PDF/A) Hide or show an annotation or form field: hide, namelist
<b>ImportData</b>	(Not for PDF/A) Import form field values from a file.
<b>JavaScript</b>	(Not for PDF/A) Execute a script with JavaScript code: script, scriptname
<b>Launch</b>	(Not for PDF/A) Launch an application or document: defaultdir, filename, newwindow, operation, parameters

Table 12.2 Action types

type	notes; options specific for this type (in addition to general options)
<b>Movie</b>	(Not for PDF/A) Play an external sound or movie file in a floating window or within the rectangle of a movie annotation: operation, target
<b>Named</b>	Execute an Acrobat menu item identified by its name: menuname
<b>ResetForm</b>	(Not for PDF/A) Set some or all form fields to their default values.
<b>SetOCGState</b>	(PDF 1.5) Hide or show layers: layerstate, preserveradio
<b>SubmitForm</b>	Send data to a uniform resource locator, i.e. an Internet address (submits which require basic authentication don't work in Acrobat): canonicaldate, exclude, exportmethod, submitemptyfields, url
<b>Trans</b>	(PDF 1.5) Update the display using some visual effect. This can be useful to control the display during a sequence of multiple actions: duration, transition
<b>URI</b>	Resolve a uniform resource identifier, i.e. jump to an Internet address: ismap, url

**optlist** An option list specifying properties of the action:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Options for properties of the action according to Table 12.3.

**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 actions, but each action must be generated with a separate call to *PDF\_create\_action()*. Using an action multiply for different objects is allowed. It is recommended to re-use existing handles if an action with the same options has already been created earlier.

PDF/X: Actions are prohibited in PDF/X.

PDF/A: Some actions are prohibited in PDF/A (see Table 12.2).

**Scope** *page, document*. The returned handle can be used until the next call to *PDF\_end\_document()*.

Table 12.3 Options for action properties with *PDF\_create\_action()*

option	explanation
<b>3Dview</b>	(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 <i>PDF_create_3dview()</i> .
<b>canonical-date</b>	(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
<b>defaultdir</b>	(String; Launch) Set the default directory for the launched application. This is only supported by Acrobat on Windows. Default: none
<b>destination</b>	(Option list; GoTo, GoToE, GoToR; required unless destname is supplied) Option list according to Table 12.5 defining the destination to jump to.

Table 12.3 Options for action properties with PDF\_create\_action()

option	explanation
<b>destname</b>	(Hypertext string) GoTo (required unless destination is supplied): name of a destination which has been defined with PDF_add_nameddest(). The destination can be created before or after referring to it. GoToR, GoToE (required unless destination is supplied): name of a destination in the remote or embedded document.
<b>duration</b>	(Float; Trans) Set the duration of the transition effect in seconds for the current page. Default: 1
<b>exclude</b>	(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
<b>export-method</b>	(Keyword list; SubmitForm) Controls how the field names and values are submitted. Default: fdf. <b>html, fdf, xfdf, pdf</b> In HTML, FDF, XPDF, or PDF format, respectively <b>annotfields</b> (Only for fdf) Include all annotations and fields. <b>coordinate</b> (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. <b>exclurl</b> (Only for fdf) The submitted FDF will exclude the url string. <b>getrequest</b> (Only for html and pdf) Submit using HTTP GET; otherwise HTTP POST <b>onlyuser</b> (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. <b>updates</b> (Only for fdf) Include all incremental updates contained in the underlying PDF document Example for combined options: exportmethod {fdf updates onlyuser}
<b>filename</b>	(Hypertext string) GoToR, Launch (required): name of an external (PDF or other) file or application which will be opened when the action is triggered. UNC file names must be written as \\server\volume. Since fully qualified file names (including paths) for Launch actions do not work in Acrobat 8 it is recommended to supply the directory name in the defaultdir option and only a simple file name (without directory components) in the filename option. ImportData (required): name of the external file containing forms data. GoToE: name of the root document of the target relative to the root document of the source. If this entry is absent, the source and target share the same root document.
<b>hide</b>	(Boolean; Hide) Indicates whether to hide (true) or show (false) annotations. Default: true
<b>ismap</b>	(Boolean; URI) If true, the coordinates of the mouse position will be added to the target URI when the url is resolved. Default: false
<b>layerstate</b>	(Option list; SetOCGState; required) List of pairs where each pair consists of a keyword and a layer handle. Supported keywords: <b>on</b> Activate the layer <b>off</b> Deactivate the layer <b>toggle</b> Reverse the state of the layer. If this is used preserveradio should be set to false.
<b>menuname</b>	(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. A full code sample for finding the names of other menu items can be found in the Cookbook topic interactive/acrobat_menu_items.

Table 12.3 Options for action properties with `PDF_create_action()`

<b>option</b>	<b>explanation</b>
<b>namelist</b>	<p>(List of strings; Hide; required) The names (including group names) of the annotations or fields to be hidden or shown.</p> <p>(SubmitForm) The names (including group names) of form fields to include in the submission or which to exclude, depending on the setting of the <code>exclude</code> option. Default: all fields are submitted except those whose <code>exportable</code> option is false.</p> <p>(ResetForm) The names (including group names) of form fields to include in the resetting or which to exclude, depending on the setting of the <code>exclude</code> option. Default: all fields are reset.</p>
<b>newwindow</b>	<p>(Boolean; GoToE, GoToR) 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.</p> <p>Launch: This entry is ignored if the file is not a PDF document. Default: Acrobat behaves according to the current user preference.</p>
<b>operation</b>	<p>This option is used differently for <code>type=Launch</code> and <code>type=Movie</code>:</p> <p>(Keyword; Launch) A keyword specifying the operation to be applied to the document specified in the <code>filename</code> option. This is only supported by Acrobat on Windows. If the <code>filename</code> option designates an application instead of a document, this option will be ignored and the application is launched. Supported keywords (default: open):</p> <p><b>open</b>      open a document</p> <p><b>print</b>      print a document</p> <p>(Keyword; Movie) A keyword specifying the operation to be applied to the movie or sound. Supported keywords (default: play):</p> <p><b>play</b>      Start playing the movie, using the mode specified in the movie annotation's <code>playmode</code> option. If the movie is currently paused, it is repositioned to the beginning before playing.</p> <p><b>stop</b>      Stop playing the movie.</p> <p><b>pause</b>      Pause a playing movie.</p> <p><b>resume</b>    Resume a paused movie.</p>
<b>parameters</b>	<p>(String; Launch) A parameter string to be passed to the application specified with the <code>filename</code> 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 <code>filename</code> designates a document. Default: none</p>
<b>preserve-radio</b>	<p>(Boolean; SetOCGState) If true, preserve the radio-button state relationship between layers. Default: true</p>
<b>script</b>	<p>(Hypertext string; JavaScript; required) A string containing the JavaScript code to be executed.</p>
<b>scriptname</b>	<p>(Hypertext string; JavaScript) If present, the JavaScript supplied in the <code>script</code> option will be inserted as a document-level JavaScript with the supplied name. If the same <code>scriptname</code> 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.</p>
<b>submit-emptyfields</b>	<p>(Boolean; SubmitForm; PDF 1.4) If true, all fields characterized by the <code>namelist</code> and <code>exclude</code> 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</p>
<b>target</b>	<p>(String; GoTo3DView, Movie; required) Name of the target 3D or movie annotation as specified in the <code>name</code> option of <code>PDF_create_annotation()</code>.</p>
<b>targetpath</b>	<p>(Option list; GoToE; required unless <code>filename</code> is specified) A target option list (see Table 12.4) specifying path information for the target document. Each target option list specifies one element in the full path to the target and may have nested target option lists with additional elements.</p>

Table 12.3 Options for action properties with `PDF_create_action()`

option	explanation
<b>transition</b>	(Keyword; Trans) Set the transition effect; see Table 3.7 for a list of keywords. Default: replace
<b>url</b>	(String; URI and SubmitForm; required) A Uniform Resource Locator encoded in 7-bit ASCII or EBCDIC (but only containing ASCII characters) specifying the link target (for type=URI) or the address of the script at the Web server that will process the submission (for type=SubmitForm). It can point to an arbitrary (Web or local) resource, and must start with a protocol identifier (such as <code>http://</code> ). The <code>textx/texty</code> , <code>currentx/currenty</code> , and <code>imagewidth/imageheight</code> parameters may be useful for retrieving positioning information for calculating the dimension of link rectangles.

Table 12.4 Suboptions for the `targetpath` option of `PDF_create_action()`

option	explanation
<b>annotation</b>	(Hypertext string; required if <code>relation=child</code> and the target is associated with a file attachment annotation) Specifies the name of the target's file attachment annotation on the page specified by <code>pagenumber</code> or <code>destname</code> .
<b>destname</b>	(Hypertext string; required unless <code>pagenumber</code> is supplied and <code>relation=child</code> and the target is associated with a file attachment annotation) Specifies a named destination for a page in the current document which contains the target's file attachment annotation. This option will be ignored if <code>pagenumber</code> is specified.
<b>name</b>	(Hypertext string; required if <code>relation=child</code> and the target is located in the attachments list; otherwise it must be absent; will be ignored if <code>annotation</code> is specified) Name of the target in the attachments list of <code>PDF_begin/end_document()</code> .
<b>pagenumber</b>	(Integer; required unless <code>destname</code> is supplied and <code>relation=child</code> and the target is associated with a file attachment annotation; will be ignored if <code>destname</code> is specified) Specifies the number of a page in the current document which contains the target's file attachment annotation.
<b>relation</b>	(Keyword; required) Specifies the relationship of the current document and the target (which may be an intermediate target). Supported keywords: <b>parent</b> The target is the parent of the current document. <b>child</b> The target is a child of the current document.
<b>targetpath</b>	(Option list) A target option list according to Table 12.4 specifying additional path information to the target document. If this option is absent the current document is the target file containing the destination.

## 12.3 Named Destinations

**C++ Java** `void add_nameddest(String name, String optlist)`

**Perl PHP** `add_nameddest(string name, string optlist)`

**C** `void PDF_add_nameddest(PDF *p, const char *name, int len, const char *optlist)`

Create a named destination on a page in the 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. An empty list is equivalent to `{type=fitwindow page=0}`. The following options can be used:

- ▶ General options: *errorpolicy* (see Table 2.6), *hypertextencoding* and *hypertextformat* (see Table 12.1)
- ▶ Destination control options according to Table 12.5:  
*bottom*, *group*, *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 12.5 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
<b>bottom</b>	(Float; only for type=fitrect) The y coordinate of the page which will positioned at the bottom edge of the window. Default: 0
<b>group</b>	(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.
<b>left</b>	(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
<b>page</b>	(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. Default: 0
<b>right</b>	(Float; only for type=fitrect) The x coordinate of the page which will positioned at the right edge of the window. Default: 1000
<b>top</b>	(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

Table 12.5 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
<b>type</b>	(Keyword) Specifies the location of the window on the target page. Supported keywords (default: <code>fitwindow</code> ):
	<b>fitheight</b> Fit the page height to the window, with the x coordinate left at the left edge of the window.
	<b>fitrect</b> Fit the rectangle specified by left, bottom, right, and top to the window.
	<b>fitvisible</b> Fit the visible contents of the page (the <code>ArtBox</code> ) to the window.
	<b>fitvisibleheight</b> Fit the visible contents of the page to the window with the x coordinate left at the left edge of the window.
	<b>fitvisiblewidth</b> Fit the visible contents of the page to the window with the y coordinate top at the top edge of the window.
	<b>fitwidth</b> Fit the page width to the window, with the y coordinate top at the top edge of the window.
	<b>fitwindow</b> Fit the complete page to the window.
	<b>fixed</b> 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.
<b>zoom</b>	(Float or percentage; only for <code>type=fixed</code> ) 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.

## 12.4 Annotations

---

```
C++ Java void create_annotation(double llx, double lly, double urx, double ury, String type, String optlist)
Perl PHP create_annotation(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 an 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, *llx/lly/urx/ury* will be ignored.

**type** The annotation type according to Table 12.6. Markup annotations are marked in the table since certain options apply only to markup annotations.

Table 12.6 Annotation types

<b>type</b>	<b>notes; options specific for this type (in addition to general options)</b>
<b>3D</b>	(PDF 1.6) <i>animated 3D model</i> : 3Dactivate, 3Ddata, 3Dinteractive, 3Dshared, 3Dinitialview
<b>Circle<sup>1</sup></b>	cloudy, createrichtext, inreplyto, interiorcolor, replyto
<b>File-Attachment<sup>1</sup></b>	(Not for PDF/A and PDF/X-1a/3) calloutline, createrichtext, filename, iconname, inreplyto, mimetype, replyto
<b>FreeText<sup>1</sup></b>	alignment, calloutline, cloudy, createrichtext, endingstyles, fillcolor, font, fontsize, inreplyto, orientate, replyto
<b>Highlight<sup>1</sup></b>	createrichtext, inreplyto, polylinelist, replyto
<b>Ink<sup>1</sup></b>	createrichtext, inreplyto, polylinelist, replyto
<b>Line<sup>1</sup></b>	captionoffset, captionposition, createrichtext, endingstyles, interiorcolor, inreplyto, leaderlength, leaderoffset, line, showcaption, replyto
<b>Link</b>	destination, destname, highlight
<b>Movie</b>	(Movie or sound annotation; not for PDF/A) filename, movieposter, playmode, showcontrols, soundvolume, windowposition, windowyscale
<b>Polygon<sup>1</sup></b>	(PDF 1.5; <i>vertices connected by straight lines</i> ): cloudy, createrichtext, inreplyto, polylinelist, replyto
<b>PolyLine<sup>1</sup></b>	(PDF 1.5; <i>similar to polygons, except that the first and last vertices are not connected</i> ) createrichtext, endingstyles, inreplyto, interiorcolor, polylinelist, replyto
<b>Popup</b>	open, parentname
<b>Square<sup>1</sup></b>	cloudy, createrichtext, inreplyto, interiorcolor, replyto
<b>Squiggly<sup>1</sup></b>	(PDF 1.4; <i>squiggly-underline annotation</i> ) createrichtext, inreplyto, polylinelist, replyto
<b>Stamp<sup>1</sup></b>	createrichtext, iconname, inreplyto, orientate, replyto

Table 12.6 Annotation types

type	notes; options specific for this type (in addition to general options)
<b>StrikeOut</b> <sup>1</sup>	createrichtext, inreplyto, polylinelist, replyto
<b>Text</b> <sup>1</sup>	(In Acrobat this type is called note annotation) createrichtext, iconname, inreplyto, open, replyto, state, statemodel
<b>Underline</b> <sup>1</sup>	createrichtext, inreplyto, polylinelist, replyto

1. Markup annotation; this is relevant for the createrichtext option.

**optlist** An option list specifying annotation properties. The following options can be used:

- ▶ General option: *hypertextencoding* (see Table 12.1)
- ▶ The following common options according to Table 12.7 are supported for all annotation types:  
*action, annotcolor, borderstyle, cloudy, contents, createdate, custom, dasharray, display, layer, linewidth, locked, lockedcontents, name, opacity, popup, readonly, rotate, subject, template, title, usematchbox, usercoordinates, zoom*
- ▶ The following type-specific options according to Table 12.7 are supported only for some annotation types according to Table 12.6:  
*3Dactivate, 3Ddata, 3Dinteractive, 3Dshared, 3Dinitialview, alignment, calloutline, caption-offset, captionposition, createrichtext, destname, endingstyles, filename, fillcolor, font, fontsize, highlight, iconname, inreplyto, interiorcolor, leaderlength, leaderoffset, line, mimetype, movieposter, open, orientate, parentname, playmode, polylinelist, replyto, showcaption, showcontrols, soundvolume, windowposition, windowyscale*

**Details** PDF/X: Annotations are only allowed if they are positioned completely outside of the BleedBox (or TrimBox/ArtBox if no BleedBox is present).

PDF/A: some annotation types and options are restricted, see Table 12.6 and Table 12.7.

Tagged PDF: the annotation will be inserted as a child of the current item if an item is currently active.

Scope page

Table 12.7 Options for PDF\_create\_annotation()

option	explanation
<b>3Dactivate</b>	(Option list; only for type=3D) Specifies when the 3D annotation should be activated and its state upon activation/deactivation. Supported options are listed in Table 12.8.
<b>3Ddata</b>	(3D handle; only for type=3D; required) 3D handle created with PDF_load_3ddata().
<b>3Dinteractive</b>	(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
<b>3Dshared</b>	(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
<b>3Dinitialview</b>	(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 views option of PDF_load_3ddata()), or default (referring to the model's defaultview option), or a 3D view handle created with PDF_create_3dview(). Default: default

Table 12.7 Options for PDF\_create\_annotation()

<b>option</b>	<b>explanation</b>
<b>action</b>	<p>(Action list) List of annotation actions for the following events (default: empty list). All types of actions are permitted:</p> <p><b>activate</b> (Only for type=Link) Actions to be performed when the annotation is activated.</p> <p><b>close</b> (PDF 1.5) Actions to be performed when the page containing the annotation is closed.</p> <p><b>open</b> (PDF 1.5) Actions to be performed when the page containing the annotation is opened.</p> <p><b>invisible</b> (PDF 1.5) Actions to be performed when the page containing the annotation is no longer visible.</p> <p><b>visible</b> (PDF 1.5) Actions to be performed when the page containing the annotation becomes visible.</p>
<b>alignment</b>	<p>(Keyword; only for type=FreeText) Alignment of text in the annotation: left, center, or right. Default: left</p>
<b>annotcolor</b>	<p>(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 (not for type=Square, Circle), gray, rgb, and (in PDF 1.6) cmyk.</p> <p>In PDF/A mode this option must only be used if an RGB output intent has been specified, and gray or rgb color must be used.</p> <p>Default: white for type=Square, Circle, otherwise none</p>
<b>borderstyle</b>	<p>(Keyword) Style of the annotation border or the line of the annotation types Polygon, PolyLine, Line, Square, Circle, Ink: solid, beveled, dashed, inset, or underline. Note that the beveled, inset, and underline styles do not work reliably in Acrobat. Default: solid</p>
<b>calloutline</b>	<p>(List of four or six floats; PDF 1.6; only for type=FreeText) List of 4 or 6 float values specifying a callout line attached to the FreeText annotation. Six numbers {x1 y1 x2 y2 x3 y3} represent the starting, knee point, and end coordinates of the line. Four numbers {x1 y1 x2 y2} represent the starting and end coordinates of the line. The coordinates will be interpreted in default coordinates (if the usercoordinates option is false) or user coordinates (if it is true).</p> <p>The start point will be decorated with the symbol specified in the first keyword of the endingstyles option.</p>
<b>captionoffset</b>	<p>(2 Floats; only for type=Line; PDF 1.7) The offset of the caption text from its normal position. The first value specifies the horizontal offset along the annotation line from its midpoint, with a positive value indicating offset to the right and a negative value indicating offset to the left. The second value specifies the vertical offset perpendicular to the annotation line, with a positive value indicating a shift up and a negative value indicating a shift down. Default: {0, 0}, i.e. no offset from the normal position</p>
<b>caption-position</b>	<p>(Keyword; only for type=Line; PDF 1.7) The annotation's caption position. This option will be ignored if showcaption=false. Supported keywords (default: Inline):</p> <p><b>Inline</b> The caption will be centered inside the line.</p> <p><b>Top</b> The caption will be positioned on top of the line.</p>
<b>cloudy</b>	<p>(Float; only for type=Circle, FreeText, Polygon, and Square; 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</p>
<b>contents</b>	<p>(String for type=FreeText, otherwise Hypertext string with a maximum length of 65535 bytes; recommended for PDF/A-1a) 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.</p>
<b>createdate</b>	<p>(Boolean; PDF 1.5) If true, a date/time entry will be created for the annotation. Default: true for Markup annotations, false otherwise</p>

Table 12.7 Options for PDF\_create\_annotation()

option	explanation
<b>createrich-text</b>	<p>(Option list; only for markup annotations; option contents must be empty; PDF 1.5) Create rich text contents from a Textflow. This may be useful to generate formatted text for annotations. Supported suboptions:</p> <p><b>textflow</b> (Textflow handle) A Textflow which will be attached to the annotation as rich text. If the Textflow handle has been supplied to PDF_fit_textflow/table() before the call to PDF_create_annotation() only the remaining text will be used for the annotation. If no more text is available the Textflow will be restarted from the beginning. Using a Textflow for an annotation does not affect subsequent calls to PDF_fit_textflow/table().</p> <p><b>userunit</b> (Keyword) Measurement unit for scalar values (e.g. firstlinedist, fontsize): cm (centimeter), in (inches), mm (millimeters), or pt (points). Default: pt</p> <p>The following Textflow options will be honored when creating rich text; all others will be ignored: nextline, alignment, fillcolor, underline, strikeout, font, fontsize, textrise, text formatting options</p> <p>Note: setting the font and alignment doesn't have the expected effect in Acrobat.</p>
<b>custom</b>	<p>(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. Supported suboptions:</p> <p><b>key</b> (String; required) 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.</p> <p><b>type</b> (Keyword; required) Type of the corresponding value, which must be one of boolean, name, or string</p> <p><b>value</b> (Hypertext string if type=string, otherwise string; required) Value as it will appear in the PDF output; PDFlib will automatically apply any decoration required for strings and names.</p>
<b>dasharray</b>	<p>(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</p>
<b>destination</b>	<p>(Option list; only for type=Link; will be ignored if an activate action has been specified) Option list according to Table 12.5 defining the destination to jump to</p>
<b>destname</b>	<p>(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(). Actions created with the destination or destname options of PDF_create_action() are dominant over this option.</p>
<b>display</b>	<p>(Keyword) Visibility on screen and paper: visible, hidden, noview, noprint. Default: visible</p>
<b>endingstyles</b>	<p>(Keyword list; only for type=FreeText, Line, PolyLine) A list with two keywords specifying the line ending styles. The second keyword will be ignored for type=FreeText (default: {none none}): none, square, circle, diamond, openarrow, closedarrow</p> <p>Additionally for PDF 1.5: butt, ropenarrow, rclosedarrow</p> <p>Additionally for PDF 1.6: slash</p>
<b>filename</b>	<p>(String; only for type=FileAttachment and Movie; required) The file name will be interpreted according to the global filenamehandling option or parameter, see Table 2.2.</p> <p>For type=FileAttachment: name of the file associated with the annotation.</p> <p>For type=Movie: name of the media file in one of the following formats: AVI or QuickTime movie, WAV or AIFF sound</p>
<b>fillcolor</b>	<p>(Color; only for type=FreeText) Fill color of the text. Supported color spaces are gray, rgb, cmyk. 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. Default: {gray 0} (=black)</p>

Table 12.7 Options for PDF\_create\_annotation()

option	explanation
<b>font</b>	(Font handle; only for type=FreeText; required) Specifies the font to be used for the annotation.
<b>fontsize</b>	(FontSize; only for type=FreeText; required) The font size in default or user coordinates depending on the usercoordinates option or parameter. The value 0 or keyword auto which means that Acrobat will adjust the fontsize to the rectangle.
<b>highlight</b>	(Keyword; only for type=Link) Highlight mode of the annotation when the user clicks on it: none, invert, outline, push. Default: invert
<b>iconname</b>	<p>(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):</p> <p>For type=Text (default: note):</p> <p>comment , key , note , help , newparagraph , paragraph , insert </p> <p>For type=Stamp, but note that these don't work reliably in Adobe Reader; the template option is recommended instead (default: draft):</p> <p>approved, experimental, notapproved, asis, expired, notforpublicrelease, confidential, final, sold, departmental, forcomment, topsecret, draft, forpublicrelease</p> <p>For type=FileAttachment (default: pushpin):</p> <p>graph , pushpin , paperclip , tag </p> <p>The template option can be used to create custom icons.</p>
<b>inreplyto</b>	(Hypertext string; PDF 1.5; only for markup annotations; required if the replyto option is supplied) The name of the annotation (see option name) that this annotation is in reply to. Both annotations must be on the same page of the document. The relationship between the two annotations must be specified by the replyto option.
<b>interiorcolor</b>	(Color; only for type=Line, PolyLine, Square, Circle) The color for the annotation's line endings, rectangle, or ellipse, respectively. Supported color spaces are none, gray, rgb, and (in PDF 1.6) cmyk. In PDF/A mode this option must only be used if an RGB output intent has been specified, and gray or rgb color must be used. Default: none
<b>layer</b>	(Layer handle; PDF 1.5) Layer to which the annotation will belong. The annotation will only be visible if the corresponding layer is visible.
<b>leaderlength</b>	<p>(List with one or two floats; the second float must not be negative; only for type=Line; PDF 1.6) The length of leader lines in default coordinates (if the usercoordinates option is false) or user coordinates (if it is true). The length is specified by two numbers (default: {0 0}):</p> <p>The first number is the extension from each endpoint of the line perpendicular to the line itself. A positive value means that the leader lines appear in the direction that is clockwise when traversing the line from its start point to its end point (as specified by the line option); a negative value indicates the opposite direction.</p> <p>The second value, which can be omitted, represents the length of leader line extensions that extend from the line proper 180° from the leader lines. A positive value will be ignored if the first value is 0.</p>
<b>leaderoffset</b>	(Non-negative float; only for type=Line; PDF 1.7) The length of the leader line offset, which is the amount of empty space between the endpoints of the annotation and the beginning of the leader lines in default coordinates (if the usercoordinates option is false) or user coordinates (if it is true). Default: 0
<b>line</b>	(Line; 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 option or parameter is false) or user coordinates (if it is true).
<b>linewidth</b>	(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

Table 12.7 Options for PDF\_create\_annotation()

option	explanation
<b>locked</b>	(Boolean) If true, the annotation properties (e.g. position and size) cannot be edited in Acrobat. However, the contents can still be modified. Default: false
<b>locked-contents</b>	(Boolean; PDF 1.7) If true, the annotation contents cannot be edited in Acrobat. However, the annotation can still be deleted or properties changed (e.g. position and size) Default: false
<b>mimetype</b>	(String; only for type=FileAttachment) MIME type of the file. Acrobat will use it for launching the appropriate application when the annotation is activated.
<b>movieposter</b>	(Keyword; only for type=Movie) Keyword which specifies a poster image representing the movie on the page. Supported keywords: auto (the poster image will be retrieved from the movie file), none (no poster will be displayed). Default: none
<b>name</b>	(Hypertext string) Name uniquely identifying the annotation. The name is necessary for some actions, and must be unique on the page. Default: none
<b>opacity</b>	(Float or percentage; PDF 1.4; not allowed for PDF/A) The constant opacity value (0-1 or 0%-100%) to be used in painting the annotation. Default: 1
<b>open</b>	(Boolean; only for type=Text, PopUp) If true, the annotation will initially be open. Default: false
<b>orientate</b>	(Keyword; only for type=FreeText, Stamp) Specifies the desired orientation of the annotation within its rectangle. Supported keywords: north (upright), east (pointing to the right), south (upside down), west (pointing to the left). Default: north
<b>parentname</b>	(String; only for type=PopUp) Name of the parent annotation for the annotation
<b>playmode</b>	(Keyword; only for type=Movie) The mode for playing the movie or sound. Supported keywords: once (play once and stop), open (play and leave the movie controller bar open), repeat (play repeatedly from beginning to end until stopped), palindrome (play continuously forward and backward until stopped). Default: once
<b>polylinelist</b>	(List containing one or more polylines or quadrilaterals; 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. <b>type=Polygon, PolyLine, Ink</b> A single list containing a polyline with n segments (minimum: n=2). <b>others</b> 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 quadrilaterals must be provided in zigzag order (top right, top left, lower right, lower left).
<b>popup</b>	(String) Name of a PopUp annotation for entering or editing the text associated with this annotation. Default: none
<b>readonly</b>	(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
<b>replyto</b>	(Keyword; PDF 1.6; only for markup annotations) Specifies the relationship (the reply type) between this annotation and the one specified by the inreplyto option. Supported keywords (default: reply): <b>reply</b> The annotation must be considered a reply to the annotation specified by inreplyto. <b>group</b> The annotation must be grouped with the annotation specified by inreplyto.
<b>rotate</b>	(Boolean; must not be set to true for text annotations 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 for text annotations in PDF/A mode, true otherwise

Table 12.7 Options for PDF\_create\_annotation()

option	explanation
<b>showcaption</b>	(Boolean; only for type=Line; PDF 1.6) If true, the text specified in the contents or createrichtext options will be replicated as a caption in the appearance of the line. Default: false
<b>showcontrols</b>	(Boolean; only for type=Movie) If true a controller bar while playing the movie or sound will be displayed. Default: false
<b>soundvolume</b>	(Float; only for type=Movie) The initial sound volume at which to play the movie, in the range -1.0 to 1.0. Higher values denote greater volume; negative values mute the sound. Default: 1.0
<b>subject</b>	(Hypertext string; PDF 1.5) Text representing a short description of the subject being addressed by the annotation. Default: none
<b>template</b>	<p>(Option list) Visual appearance of the annotation for one or more states. This may be useful e.g. for custom stamps. Supported suboptions:</p> <p><b>normal/rollover/down</b>            (Template handle or keyword) Template which will be used for the annotation's normal, mouse rollover, or mouse button down appearance, respectively. The keyword viewer instructs Acrobat to create the appearance when rendering the page. Default for normal: viewer; default for rollover and down: the value of normal</p> <p><b>fitmethod</b> (Keyword) Method to fit the template into the annotation rectangle. If fitmethod is different from entire the annotation rectangle will be adapted to the template box (default: entire):</p> <p><b>nofit</b> Position the template only, without any scaling or clipping.</p> <p><b>meet</b> Position the template according to the position option, and scale it so that it entirely fits into the rectangle while preserving its aspect ratio. Generally at least two edges of the template will meet the corresponding edges of the rectangle.</p> <p><b>entire</b> Position the template according to the position option, and scale it such that it entirely covers the rectangle. Generally this method will distort the template.</p> <p><b>position</b> (List of floats or keywords) One or two values specifying the position of the reference point (x, y) within the template with {0 0} being the lower left corner of the template, and {100 100} the upper right corner. The values are expressed as percentages of the template's 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}.</p>
<b>title</b>	(Hypertext string; recommended for movie annotations) The text label to be displayed in the title bar of the annotation's pop-up window when open and active. This string corresponds to the »Author« field in Acrobat. The maximum length of title is 255 single-byte characters or 126 Unicode characters. However, a practical limit of 32 characters is advised. Default: none
<b>usematchbox</b>	<p>(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.</p> <p>If the matchbox itself or the specified rectangle does not exist on the current page, the function will silently return without creating any annotation.</p>
<b>user-coordinates</b>	(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
<b>window-position</b>	(List of 2 floats or keywords; only for type=Movie) For floating movie windows, the relative position of the window on the screen. The two values specify the position of the floating window on the screen, with {0 0} denoting the lower left corner of the screen and {100 100} the upper right corner. The keywords left, center, right (in horizontal screen direction) or bottom, center, top (in vertical screen direction) can be used as equivalents for the values 0, 50, and 100. Default: {center center}

Table 12.7 Options for PDF\_create\_annotation()

option	explanation
<b>windowyscale</b>	(Float or keyword; only for type=Movie) The zoom factor at which to play the movie in a floating window. If the option is absent, the movie will be played in the annotation rectangle. The value of this option is either a zoom factor for the movie, or the following keyword (default: option is absent, i.e. the movie is played in the annotation rectangle): <b>fullscreen</b> The movie will be played using all of the available screen area.
<b>zoom</b>	(Boolean; must not be set to true for text annotations 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 for text annotations in PDF/A mode, true otherwise

Table 12.8 Suboptions for the 3Dactivate option of PDF\_create\_annotation()

option	explanation
<b>enable</b>	(Keyword) Specifies when the animation should be enabled (default: click): <b>open</b> Activate when the page is opened. <b>visible</b> Activate when the page becomes visible. <b>click</b> Annotation must explicitly be activated by a script or user action.
<b>enablestate</b>	(Keyword) Initial animation state (default: play): <b>pause</b> The 3D model is instantiated, but script animations are disabled. <b>play</b> The 3D model is instantiated; script animations are enabled if present.
<b>disable</b>	(Keyword) Specifies when the animation should be disabled (default: invisible): <b>close</b> Deactivate when the page is closed. <b>invisible</b> Deactivate when the page becomes invisible. <b>click</b> Annotation must explicitly be deactivated by a script or user action.
<b>disablestate</b>	(Keyword) State of the animation upon disabling (default: reset): <b>pause</b> The 3D model can be rendered, but animations are disabled. <b>play</b> The 3D model can be rendered and animations are enabled. <b>reset</b> Initial state of the 3D model before it has been used in any way.
<b>modeltree</b>	(Boolean; PDF 1.6) If true, the Model Tree navigation tab will be opened when the annotation is activated (default: false)
<b>toolbar</b>	(Boolean; PDF 1.6) If true, the 3D toolbar (at the top of the annotation) will be displayed when the annotation is activated (default: true)

## 12.5 Form Fields

*Cookbook* A full code sample can be found in the *Cookbook* topic `webserver/starter_webform`.

---

```

C++ Java void create_field(double llx, double lly, double urx, double ury,
        String name, String type, String optlist)
Perl PHP create_field(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 12.9.

Table 12.9 Form field types

type	icon	Options specific for this type (in addition to general options)
<b>pushbutton</b>		buttonlayout, caption, captiondown, captionrollover, charspacing, fitmethod, icon, icondown, iconrollover, position, submitname
<b>checkbox</b>		currentvalue, itemname
<b>radiobutton</b>		buttonstyle, currentvalue, itemname, toggle, unisonselect <i>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.</i>
<b>listbox</b>		charspacing, currentvalue, itemnamelist, itemtextlist, multiselect, sorted, topindex
<b>combobox</b>		commitonselect, charspacing, currentvalue, editable, itemnamelist, itemtextlist, sorted, spellcheck
<b>textfield</b>		comb, charspacing, currentvalue, fileselect, maxchar, multiline, password, richtext, scrollable, spellcheck
		<i>Text fields are also used for barcodes:</i> barcode
<b>signature</b>		charspacing, lockmode

**optlist** An option list specifying field properties:

- ▶ General options: *errorpolicy* (see Table 2.6), *hypertextencoding* and *hypertextformat* (see Table 12.1)
- ▶ An option list specifying the field's properties according to Table 12.10. 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 12.9 for more type-specific options):  
*action, alignment, backgroundcolor, barcode, bordercolor, borderstyle, calorder, dasharray, defaultvalue, display, exportable, fieldtype, fillcolor, font, fontsize, highlight, 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*

---

**C++ Java** `void create_fieldgroup(String name, String optlist)`  
**Perl PHP** `create_fieldgroup(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 with field options for *PDF\_create\_field()*

**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*

Table 12.10 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

<b>option</b>	<b>explanation</b>
<b>action</b>	<p>(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, radiobutton. Default: empty list</p> <p><b>activate</b> Actions to be performed when the field is activated.</p> <p><b>blur</b> Actions to be performed when the field loses the input focus.</p> <p><b>calculate</b> JavaScript actions to be performed in order to recalculate the value of this field when the value of another field changes.</p> <p><b>close</b> (PDF 1.5) Actions to be performed when the page containing the field is closed.</p> <p><b>down</b> Actions to be performed when the mouse button is pressed inside the field's area.</p> <p><b>enter</b> Actions to be performed when the mouse enters the field's area.</p> <p><b>exit</b> Actions to be performed when the mouse exits the field's area.</p> <p><b>focus</b> Actions to be performed when the field receives the input focus.</p> <p><b>format</b> 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.</p> <p><b>invisible</b> (PDF 1.5) Actions to be performed when the page containing the field is no longer visible.</p> <p><b>keystroke</b> 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.</p> <p><b>open</b> (PDF 1.5) Actions to be performed when the page containing the field is opened.</p> <p><b>up</b> Actions to be performed when the mouse button is released inside the field's area (this is typically used to activate a field).</p> <p><b>validate</b> JavaScript actions to be performed when the field's value is changed. This allows the new value to be checked for validity.</p> <p><b>visible</b> (PDF 1.5) Actions to be performed when the page containing the field becomes visible.</p>
<b>alignment</b>	(Keyword) Alignment of text in the field: left, center, right. Default: left
<b>background-color</b>	(Color) Color of the field background or border. Supported color spaces: none, gray, rgb, cmyk. Default: none
<b>bordercolor</b>	
<b>barcode</b>	<p>(Option list; only for type=textfield; implies readonly; PDF 1.7ext3) Create a barcode field according to the options in Table 12.11. The field should provide the action option with a calculate event script which determines the barcode contents based on the contents of other fields or supplies a static value: <code>action={calculate=...}</code>.</p> <p>The barcode will be rendered in Acrobat 9 or above, but not any version of Adobe Reader. Acrobat 9 and X will crash if the first field on a page is a barcode field. In order to work around this problem you must create another field before adding the barcode field. The first field may be as simple as a dummy text field with zero width and height to prevent the crash.</p>
<b>borderstyle</b>	(Keyword) Style of the field border, which is one of solid, beveled, dashed, inset, underline. Default: solid
<b>button-layout</b>	(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
<b>buttonstyle</b>	(Keyword; only for type=radiobutton and checkbox) Specifies the symbol to be used for the field: check, cross, diamond, circle, star, square. Default: check
<b>calcorder</b>	(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)

Table 12.10 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>caption</b>	(Content string; only for <code>type=pushbutton</code> ; one of the <code>caption</code> or <code>icon</code> options must be supplied for push buttons) The caption text which will be visible when the button doesn't have input focus. It will be displayed with the font supplied in the <code>font</code> option. Use an empty string (i.e. <code>caption { }</code> ) if you don't want caption or icon. Default: none
<b>caption-down</b>	(Content string; only for <code>type=pushbutton</code> ) The caption text which will be visible when the button is activated. It will be displayed with the font supplied in the <code>font</code> option. Default: none
<b>caption-rollover</b>	(Content string; only for <code>type=pushbutton</code> ) The caption text which will be visible when the button has input focus. It will be displayed with the font supplied in the <code>font</code> option. Default: none
<b>charspacing</b>	(Float; not for <code>type=radiobutton</code> , <code>checkbox</code> ) 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
<b>comb</b>	(Boolean; only for <code>type=textfield</code> ; PDF 1.5) If true and the <code>multiline</code> , <code>fileselect</code> , and <code>password</code> options are false, and the <code>maxchar</code> option has been supplied with an integer value, the field will be divided into a number of equidistant subfields (according to the <code>maxchar</code> option) for individual characters. Default: false
<b>commit-onselect</b>	(Boolean; only for <code>type=listbox</code> , <code>combobox</code> ; 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
<b>currentvalue</b>	(Not for <code>type=pushbutton</code> , <code>signature</code> ) The field's initial value. Type and default depend on the field type: <b>checkbox, radiobutton</b> (String) Arbitrary string other than <code>Off</code> means that the button is activated. The string <code>Off</code> means that the button is deactivated. This option should be set for the first button. Default: <code>Off</code> <b>textfield, combobox</b> (Content string) Contents of the field. It will be displayed with the font supplied in the <code>font</code> option. Default: empty <b>listbox</b> (List of integers) Indices of the selected items within <code>itemtextlist</code> . Default: none
<b>dasharray</b>	(List of floats; only for <code>borderstyle=dashed</code> ). The lengths of dashes and gaps for a dashed border in default units (see <code>PDF_setdash()</code> ). Default: 3 3
<b>defaultvalue</b>	The field's value after a reset action. Types and defaults are the same as for the <code>currentvalue</code> option. Exception: for listboxes only a single integer value is allowed.
<b>display</b>	(Keyword) Visibility on screen and paper: <code>visible</code> , <code>hidden</code> , <code>noview</code> , <code>noprint</code> . Default: <code>visible</code>
<b>editable</b>	(Boolean; only for <code>type=combobox</code> ) If true, the currently selected text in the box can be edited. Default: false
<b>exportable</b>	(Boolean) The field will be exported when a <code>SubmitForm</code> action happens. Default: true
<b>fieldtype</b>	(Keyword; only for <code>PDF_create_fieldgroup()</code> ) Type of the fields contained in the group: <code>mixed</code> , <code>pushbutton</code> , <code>checkbox</code> , <code>radiobutton</code> , <code>listbox</code> , <code>combobox</code> , <code>textfield</code> , or <code>signature</code> . Unless <code>fieldtype=mixed</code> the group may only contain fields of the specified type. If a particular <code>fieldtype</code> 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 <code>fieldtype=radiobutton</code> the option <code>unisonselect</code> must be supplied. The options <code>itemtextlist</code> , <code>itemnamelist</code> , <code>currentvalue</code> and <code>defaultvalue</code> must be specified in the options of <code>PDF_create_fieldgroup()</code> , and not in the options of <code>PDF_create_field()</code> . Default: <code>mixed</code>
<b>fileselect</b>	(Boolean; only for <code>type=textfield</code> ) If true, the text in the field will be treated as a file name. Default: false
<b>fillcolor</b>	(Color) Fill color for text. Supported color spaces: <code>gray</code> , <code>rgb</code> , <code>cmyp</code> . Default: <code>{gray 0}</code> (=black)

Table 12.10 Options for field properties with PDF\_create\_field() and PDF\_create\_fieldgroup()

<b>option</b>	<b>explanation</b>
<b>fitmethod</b>	(Keyword; only for type=pushbutton) Method of placing a template provided with the icon, icondown, and iconrollover options within the button. Supported keywords (default: meet): <b>auto</b> same as meet if the template fits into the button, otherwise clip <b>nofit</b> same as clip <b>clip</b> template will not be scaled, but clipped at the field border <b>meet</b> template will be scaled proportionally so that it fits into the button <b>slice</b> same as meet <b>entire</b> template will be scaled so that it fully fits into the button
<b>font</b>	(Font handle; required except for type=radiobutton and checkbox which always use ZapfDingbats; for type=pushbutton it is only required if one or more of the caption, captionrollover, or captiondown options are specified). The font to be used for the field. The following font types are not allowed for text form fields: CJK fonts with legacy CMaps, TrueType or OpenType subsets (i.e. subsetting=true), CID fonts (i.e. autocidfont=true or encoding=unicode for TrueType fonts). Acrobat can display characters even if they are not included in the font's encoding. For example, you can use encoding=winansi and supply Unicode characters outside winansi.
<b>fontsize</b>	(FontSize) Font size in user coordinates. The value 0 or keyword auto which means that Acrobat will adjust the fontsize to the rectangle. Default: auto
<b>highlight</b>	(Keyword) Highlight mode of the field when the user clicks on it: none, invert, outline, push. Default: invert
<b>icon</b>	(Template handle <sup>1</sup> ; 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
<b>icondown</b>	(Template handle <sup>1</sup> ; only for type=pushbutton) Handle for a template which will be visible when the button is activated. Default: none
<b>iconrollover</b>	(Template handle <sup>1</sup> ; only for type=pushbutton) Handle for a template which will be visible when the button has input focus. Default: none
<b>itemname</b>	(Hypertext string; only for type=radiobutton, 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. Default: field name
<b>item-namelist</b>	(Hypertext string; only for type=listbox, combobox) Export values of the list items. Multiple items may have the same export value. Default: none
<b>itemtextlist</b>	(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.
<b>layer</b>	(Layer handle; PDF 1.5) Layer to which the field will belong. The field will only be visible if the corresponding layer is visible.
<b>linewidth</b>	(Integer) Line width of the field border in default units (=points). Default: 1
<b>locked</b>	(Boolean) If true, the field properties cannot be edited in Acrobat. Default: false
<b>lockmode</b>	(Keyword; only for type=signature; PDF 1.5) Indicates the set of fields that should be locked when the field is signed: <b>all</b> All fields in the document will be locked.
<b>maxchar</b>	(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

Table 12.10 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>multiline</b>	(Boolean; only for type=textfield) If true, text will be wrapped to multiple lines if required. Default: false
<b>multiselect</b>	(Boolean; only for type=listbox) If true, multiple items in the list can be selected. Default: false
<b>orientate</b>	(Keyword) Orientation of the contents within the field: north, west, south, east. Default: north
<b>password</b>	(Boolean; only for type=textfield) If true, the text will be simulated with bullets or asterisks upon input. Default: false
<b>position</b>	(List of floats or keywords; only for type=pushbutton) One or two values specifying the position of a template provided with the icon... options within the field rectangle, with {0 0} being the lower left corner of the field, and {100 100} the upper right corner. The values are expressed as percentages of the field rectangle's 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: {center}. Examples: {0 50} or {left center}      left-justified template {50 50} or {center}          centered template {100 50} or {right center} right-justified template
<b>readonly<sup>2</sup></b>	(Boolean) If true, the field does not allow any input. Default: false
<b>required</b>	(Boolean) If true, the field must contain a value when the form is submitted. Default: false
<b>richtext</b>	(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
<b>scrollable</b>	(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
<b>sorted</b>	(Boolean; only for type=listbox and combobox) If true, the contents of the list will be sorted. Default: false
<b>spellcheck</b>	(Boolean; only for type=textfield and combobox) If true, the spell checker will be active in the field. Default: true
<b>strokecolor</b>	(Color) Stroke color for text. Supported color spaces: gray, rgb, cmyk. Default: {gray 0} (=black).
<b>submitname</b>	(Hypertext string; recommended only for type=pushbutton) URL-encoded string of the Internet address to which the form will be submitted. Default: none
<b>taborder</b>	(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.
<b>toggle</b>	(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
<b>tooltip<sup>2</sup></b>	(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
<b>topindex</b>	(Integer; only for type=listbox) Index of the first visible entry. The first item has index 0. Default: 0
<b>unisonselect</b>	(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

Table 12.10 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>user-coordinates</b>	(Boolean) If <code>false</code> , 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 <code>usercoordinates</code> parameter

1. Templates for icons can be created with the `PDF_begin_template_ext()` 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()`.

Table 12.11 Suboptions for the barcode option of `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>caption</b>	(Hypertext string) Caption which will be rendered below the barcode. By default, Acrobat creates the file: URL for the document as caption.
<b>dataprep</b>	(Integer) Applicable data preparation. Supported values (default: 0): <b>0</b> Do not apply any compression before encoding the data in the barcode. <b>1</b> Compress the data with the Flate compression algorithm before encoding the data.
<b>ecc</b>	(Integer; required for <code>symbology=PDF417</code> and <code>QRCode</code> ) Error correction coefficient where higher values create better error correction through redundancy, but require a larger barcode. For <code>symbology=PDF417</code> the values must be in the range 0-8; for <code>symbology=QRCode</code> the values must be in the range 0-3.
<b>resolution</b>	(Positive integer) Resolution in dpi at which the barcode is rendered (default: 300)
<b>symbology</b>	(Keyword; required) Barcode technology to use: <b>PDF417</b> PDF417 bar code according to ISO 15438 <b>QRCode</b> QR Code 2005 bar code according to ISO 18004 <b>DataMatrix</b> Data Matrix bar code according to ISO 16022
<b>xsymheight</b>	(Integer; only for <code>symbology=PDF417</code> , and required in this case) Vertical distance between two barcode modules in pixels. The ratio <code>xsymheight/xsymwidth</code> must be an integer value. The allowed range for this ratio is 1-4.
<b>xsymwidth</b>	(Integer; required) Horizontal distance between two barcode modules in pixels

## 12.6 Bookmarks

---

```
C++ Java int create_bookmark(String text, String optlist)
Perl PHP int create_bookmark(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.

**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. The following options can be used:

- ▶ General options: *errorpolicy* (see Table 2.6), *hypertextencoding* and *hypertextformat* (see Table 12.1)
- ▶ Bookmark control options according to Table 12.12:  
*action*, *destination*, *destname*, *fontstyle*, *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 12.12 Options for *PDF\_create\_bookmark()*

option	explanation
<b>action</b>	(Action list) List of bookmark actions for the following event. Default: GoTo action with the target specified in the <i>destination</i> option. <b>activate</b> Actions to be performed when the bookmark is activated. All types of actions are permitted.
<b>destination</b>	(Option list; will be ignored if an <i>activate</i> action has been specified) Option list specifying the bookmark destination according to Table 12.5. Default: {type fitwindow page 0} if <i>destination</i> , <i>destname</i> , and <i>action</i> are absent.
<b>destname</b>	(Hypertext string; may be empty; will be ignored if the <i>destination</i> option has been specified) Name of a destination which has been defined with <i>PDF_add_nameddest()</i> . Destination or <i>destname</i> actions will be dominant over this option. If <i>destname</i> is an empty string (i.e. {}) and neither <i>destination</i> nor <i>action</i> are specified, the bookmark won't have any action, which may be useful if the bookmark serves as a separator.
<b>fontstyle</b>	(Keyword) Specifies the font style of the bookmark text: normal, bold, italic, bolditalic. Default: normal
<b>index</b>	(Integer) Index where to insert the bookmark within the <i>parent</i> . 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 <i>parent</i> . 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).

Table 12.12 Options for `PDF_create_bookmark()`

<b>option</b>	<b>explanation</b>
<b>open</b>	(Boolean) If <code>false</code> , subordinate bookmarks will not be visible. If <code>true</code> , all children will be folded out. Default: <code>false</code>
<b>parent</b>	(Bookmark handle) The new bookmark will be specified as a subordinate of the bookmark specified in the handle. If <code>parent=0</code> a new top-level bookmark will be created. Default: <code>0</code>
<b>textcolor</b>	(Color) Specifies the color of the bookmark text. Supported color spaces: <code>none</code> , <code>gray</code> , <code>rgb</code> . Default: <code>rgb {0 0 0}</code> (=black)

## 12.7 PDF Packages and Portfolios

---

**C++ Java** `int add_portfolio_folder(int parent, String, foldername, String optlist)`  
**Perl PHP** `int add_portfolio_folder(int parent, string foldername, string optlist)`  
**C** `int PDF_add_portfolio_folder(PDF *p, int parent, const char *foldername, int len, const char *optlist)`

---

Add a folder to a new or existing portfolio (requires PDF 1.7ext3).

**parent** The parent folder, specified by a folder handle returned by an earlier call to `PDF_add_portfolio_folder()`, or -1 (in PHP: 0) for the root folder.

**foldername** (Hypertext string with 1-255 characters; the characters / \ : \* " < > | must not be used; the last character must not be a period '.') Name of the folder. Two folders with the same parent must not have the same name after case normalization. The name of the root folder will be ignored by Acrobat.

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

**optlist** An option list specifying portfolio properties. The following options can be used:

- ▶ General options: *errorpolicy* (see Table 2.6), *hypertextencoding* and *hypertextformat* (see Table 12.1)
- ▶ Portfolio options according to Table 12.13: *description*, *fieldlist*, *thumbnail*

**Returns** A handle which can be used in `PDF_add_portfolio_folder()` or `PDF_add_portfolio_file()`.

**Details** The generated folder structure will be used to create a PDF portfolio for the current document. The folder structure will be deleted after `PDF_end_document()`. This function must not be used if the attachments option has been supplied to `PDF_begin_document()`.

**Scope** *document*

Table 12.13 Options for `PDF_add_portfolio_folder()`

<b>option</b>	<b>explanation</b>
<b>description</b>	(Hypertext string) Description of the folder
<b>fieldlist</b>	(List of option lists) Specify metadata fields for the folder. Each list refers to a field in the schema suboption of the <i>portfolio</i> option of <code>PDF_end_document()</code> . Supported suboptions are listed in Table 12.15.
<b>thumbnail</b>	(Image handle) Specifies an image to be used as thumbnail for the folder. The handle must have been created with <code>PDF_load_image()</code> and the image must satisfy the conditions listed for <code>PDF_add_thumbnail()</code> .

---

**C++ Java** `int add_portfolio_file(int folder, String filename, String optlist)`  
**Perl PHP** `int add_portfolio_file(int folder, string filename, string optlist)`  
**C** `int PDF_add_portfolio_file(PDF *p, int folder, const char *filename, int len, const char *optlist)`

---

Add a file to a portfolio folder or a package (requires PDF 1.7).

**folder** A folder handle returned by an earlier call to `PDF_add_portfolio_folder()` or -1 (in PHP: 0) for the root folder. Folders different from the root folder require PDF 1.7ext3.

**filename** (Name string; will be interpreted according to the global *filenamehandling* option or parameter, see Table 2.2) Name of a disk-based or virtual file which will be attached to the specified folder of the PDF portfolio. With the *createpvf* option of *PDF\_begin\_document()* you can create documents in memory and pass them on for inclusion in a PDF Portfolio without creating any temporary files on disk.

Note that Acrobat will use the file name suffix to determine which application to launch when interacting with the file in Acrobat. If a file name with the appropriate suffix cannot be used due to external restrictions you can create a PVF file (which supports arbitrary file names) instead.

**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 file properties:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Options for file properties according to Table 12.14:  
*description, fieldlist, mimetype, name, password, thumbnail*

**Returns** The value 1 if the file could be added successfully, or an error code of -1 (in PHP: 0) if the function call failed. If *errorpolicy=exception* this function will throw an exception in case of an error. PDF documents will be opened to fetch the modification and creation dates. If the PDF document cannot be opened (e.g. because no password was supplied) the document will be included in the PDF portfolio nevertheless.

**Details** The specified file will be attached to the specified folder of a PDF 1.7ext3 portfolio or a PDF 1.7 package. If PDI is available, PDF documents will be opened if possible and their creation and modification dates will be written to the portfolio. This function must not be used if the attachments option has been supplied to *PDF\_begin\_document()*.

**Scope** *document*

Table 12.14 Options for *PDF\_add\_portfolio\_file()*

option	explanation
<b>description</b>	(Hypertext string) Descriptive text associated with the file.
<b>fieldlist</b>	(List of option lists) Specify metadata fields for the file. Each list refers to a field in the schema suboption of the portfolio option of <i>PDF_end_document()</i> . Supported suboptions are listed in Table 12.15.
<b>mimetype</b>	(String) MIME type of the file. Note that Acrobat will use the filename suffix instead of the MIME type for launching the appropriate application when the file is activated. The MIME type application/pdf will be set automatically if the file can successfully be opened as PDF document.
<b>name</b>	(Hypertext string) Name of the file to be used in the portfolio if a name different from filename is desired. It is recommended to use names with the usual type-specific suffixes (e.g. .pdf) to make sure that Acrobat will properly preview PDF documents and launch the appropriate application for other file types. Two file names in the same folder must be different after case normalization. It is recommended to avoid slash characters '/' in the name since Acrobat will drop all characters before the slash. Default: filename without any path components.
<b>password</b>	(String with up to 127 characters; only if PDI is available) PDF master password required to open a protected PDF document for fetching its date entries.
<b>thumbnail</b>	(Image handle) Specifies an image to be used as thumbnail for the file. The handle must have been created with <i>PDF_load_image()</i> and the image must satisfy the conditions listed for <i>PDF_add_thumbnail()</i> .

Table 12.15 Suboptions of the `fieldlist` option of `PDF_add_portfolio_folder()` and `PDF_add_portfolio_file()`

<b>option</b>	<b>explanation</b>
<b>key</b>	(String; required) Name of the field, which must refer to a key in the schema suboption of the portfolio option list of <code>PDF_end_document()</code> . The name must be unique.
<b>prefix</b>	(Hypertext string) A prefix string which will be prepended to the field value presented to the user. Acrobat will use this entry only if <code>type=text</code> . Default: none
<b>type</b>	(Keyword) Data type of the field. Supported keywords (default: text): <b>text</b> Text field: the field value will be stored as hypertext string. <b>date</b> Date field: the field value will be stored as PDF date string. <b>number</b> Number field: the field value will be stored as PDF number.
<b>value</b>	(Hypertext string; required) Specifies the value of a field in the schema suboption of the portfolio option list of <code>PDF_end_document()</code> . The data type must be specified in the type option and must match the corresponding type suboption of the schema suboption of the portfolio option.

Table 12.16 Suboptions of the portfolio option of PDF\_end\_document()

option	explanation
<b>coversheet</b>	(Hypertext string) The name of the file within the portfolio which will be initially presented in the user interface. Default: the document which contains the portfolio
<b>coversheet-folder</b>	(Folder handle) The name of the folder within the portfolio which contains the file specified in the coversheet option. If a file with the coversheet name exists in multiple portfolio folders and no coversheetfolder has been specified, the first occurrence will be used. Default: none
<b>initialview</b>	(Keyword) Specifies the initial view. Supported keywords (default: detail): <ul style="list-style-type: none"> <li><b>detail</b> The portfolio is presented in details mode, with all information in the schema option presented in a multi-column format. This mode provides the most information to the user (Acrobat: »View top«).</li> <li><b>tile</b> The portfolio is presented in tile mode, with each file in the collection denoted by a small icon and a subset of information from the schema option. This mode provides top-level information about the file attachments to the user (Acrobat: »View left«).</li> <li><b>hidden</b> The portfolio is initially hidden, without preventing the user from obtaining a file list via explicit action (Acrobat: »Minimize view«).</li> </ul>
<b>schema</b>	(List of option lists) Metadata schema for the portfolio: each option list defines a field with a unique name which corresponds to a key in the fieldlist of a folder or file, or to the name of a standard field. These fields define the display behavior of the portfolio in Acrobat (default: Acrobat displays the file name and size, modification date, and description if specified): <ul style="list-style-type: none"> <li><b>editable</b> (Boolean) Specifies whether Acrobat should allow editing the field value. Default: false</li> <li><b>key</b> (String; required) The internal field name, which must be unique. The following names (which can not be used for user-defined fields) can be used to assign new labels to the builtin fields: <code>_creationdate</code>, <code>_description</code>, <code>_filename</code>, <code>_moddate</code>, <code>_size</code>.</li> <li><b>label</b> (Hypertext string; required) The textual field label that is displayed to the user.</li> <li><b>order</b> (Integer) Relative order of the fields in the user interface (1,2,3,...)</li> <li><b>type</b> (Keyword) Data type of the field. The following types can be used to refer to user-defined fields in the fieldlist option (default: text): <ul style="list-style-type: none"> <li><b>text</b> hypertext string</li> <li><b>date</b> PDF date string</li> <li><b>number</b> number</li> </ul> </li> <li><b>visible</b> (Boolean) Initial visibility of the field in the user interface. Default: true; however, in the presence of user-defined fields Acrobat will hide builtin fields unless they are explicitly specified as visible.</li> </ul>
<b>sort</b>	(List of option lists, where each list contains a string and an optional keyword) Specifies the order in which the fields specified in the schema option will be sorted in the user interface. Each sublist contains the field name (required) and a keyword (optional). Supported keywords (Default: ascending): <ul style="list-style-type: none"> <li><b>ascending</b> field values are sorted in ascending order</li> <li><b>descending</b> field values are sorted in descending order</li> </ul> <p>Acrobat uses this list to sort the fields in the portfolio. The list is used to allow additional fields to contribute to the sort, where each additional field is used to break ties: if multiple fields in the schema option have the same value for the first field in the list, the values for successive fields in the list are used for sorting until a unique order is determined or until the field names are exhausted. Default: no sorting</p>

Table 12.16 Suboptions of the portfolio option of PDF\_end\_document()

<b>option</b>	<b>explanation</b>
<b>split</b>	(Option list; PDF 1.7ext3) Specifies the orientation and position of the splitter bar. The default depends on the initialview option: The value detail (or no value) implies horizontal orientation and tile indicates vertical orientation. No splitter is used if initialview=hidden. Supported suboptions:
<b>direction</b>	(Keyword) Orientation of the splitter bar. Supported keywords: <b>horizontal</b> Split the window horizontally. <b>vertical</b> Split the window vertically. <b>none</b> Don't split the window. The entire window is dedicated to the file navigation view.
<b>position</b>	(Percentage) Initial position of the splitter bar, specified as a percentage of the available window area. Allowed values are in the range from 0 to 100. This entry will be ignored if direction=none. Default: viewer dependent



# 13 3D and Geospatial Features

## 13.1 3D Artwork

*Cookbook* A full code sample can be found in the Cookbook topic `multimedia/starter_3d`.

---

```
C++ Java int load_3ddata(String filename, String optlist)
Perl PHP int 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; will be interpreted according to the global `filenamehandling` option or parameter, see Table 2.2) Name of a disk-based or virtual file containing a 3D model.

**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:

- ▶ General option: `errorpolicy` (see Table 2.6) and `hypertextencoding` (see Table 12.1)
- ▶ Options for specifying properties of the 3D model according to Table 13.1: `defaultview`, `script`, `type`, `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. If `errorpolicy=return` the caller must check for a return value of -1 (in PHP: 0) since it signals an error.

**Details** Load a 3D model from a file containing PRC or U3D format.

**Scope** `page`, `document`. The returned handle can be used until the next call to `PDF_end_document()`.

Table 13.1 Options for `PDF_load_3ddata()`

option	explanation
<b>defaultview</b>	(Keyword or 3D view handle) Specifies the initial view of the 3D annotation; One of the keywords <code>first</code> or <code>last</code> (referring to the respective entries in the <code>views</code> option), or a 3D view handle created with <code>PDF_create_3dview()</code> . Default: <code>first</code>
<b>script</b>	(Hypertext string) String containing JavaScript code to be executed when the 3D model is instantiated. Default: no script

Table 13.1 Options for PDF\_load\_3ddata()

option	explanation
<b>type</b>	(Keyword) Specify the type of 3D data (default: U3D): <b>PRC</b> (PDF 1.7ext3) Product Representation Compact (PRC) format (ISO 14739-1) <b>U3D</b> Universal 3D File Format (U3D) in the following flavors (see www.ecma-international.org): PDF 1.6, but requires Acrobat 7.0.7 or above: ECMA-363, Universal 3D File Format (U3D), 1st Edition; PDF 1.6, but requires Acrobat 8.1 or above: ECMA-363, Universal 3D File Format (U3D), 3rd Edition; Note that Acrobat 9.3.x and 9.4.x (but not Acrobat 8 and X) have trouble displaying U3D artwork, and issue an error message »A 3D data parsing error has occurred« instead.
<b>views</b>	(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(). The type option used when creating the views with PDF_create_3dview() must match the type option in PDF_load_3ddata(). Default: empty list

**C++ Java** `int create_3dview(String username, String optlist)`

**Perl PHP** `int 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:

- ▶ General options: *errorpolicy* (see Table 2.6) and *hypertextencoding* (see Table 12.1)
- ▶ Options for specifying 3D view properties according to Table 13.2:  
*background*, *camerazworld*, *cameradistance*, *lighting*, *namerendermode*, *type*, *U3Dpath*

**Returns** A 3D view handle which can be used until the end of the enclosing *document* scope. If *errorpolicy*=*return* the caller must check for a return value of -1 (in PHP: 0) since it signals an error.

**Details** The 3D view handle can be attached to 3D models with the *views* option in *PDF\_load\_3ddata()* or can be used to create 3D annotations with *PDF\_create\_annotation()* or 3D-related actions with *PDF\_create\_action()*.

**Scope** *page*, *document*. The returned handle can be used until the next call to *PDF\_end\_document()*.

Table 13.2 Options for PDF\_create\_3dview()

option	explanation
<b>background</b>	(Option list) Specifies the background for the 3D model: <b>fillcolor</b> (Color) Background color, expressed in the RGB color space. Default: white <b>entire</b> (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
<b>camerazworld</b>	(List of 12 floats) 3D transformation matrix specifying position and orientation of the camera in world coordinates (see description below). Default: the initial view defined internally in the 3D model

Table 13.2 Options for PDF\_create\_3dview()

option	explanation
<b>camera-distance</b>	(Float; must not be negative; will be ignored if camera2world is not specified) Distance between the camera and the center of the orbit. For details see description of the CO key in section 13.6.4 »3D Views« of ISO 32000-1. Default: defined internally in the 3D data
<b>lighting</b>	(Option list; PDF 1.7) Specifies the lighting scheme for the 3D artwork. The following option is supported: <b>type</b> (Keyword) Specifies the lighting scheme. Supported keywords (Default: Artwork): <b>Artwork</b> Lights are specified in the 3D artwork. <b>None</b> No lights; lights specified in the 3D artwork will be ignored. <b>White</b> Three light-grey infinite lights, no ambient term <b>Day</b> Three light-grey infinite lights, no ambient term <b>Night</b> One yellow, one aqua, and one blue infinite light, no ambient term <b>Hard</b> Three grey infinite lights, moderate ambient term <b>Primary</b> One red, one green, and one blue infinite light, no ambient term <b>Blue</b> Three blue infinite lights, no ambient term <b>Red</b> Three red infinite lights, no ambient term <b>Cube</b> Six grey infinite lights aligned with the major axes, no ambient term <b>CAD</b> Three grey infinite lights and one light attached to the camera, no ambient term <b>Headlamp</b> Single infinite light attached to the camera, low ambient term
<b>name</b>	(Hypertext string) Name of the 3D view, which can be used in GoTo actions. This is an optional internal name which is treated separately from the required username parameter.
<b>rendermode</b>	(Option list; PDF 1.7) Specifies the render mode for displaying the 3D artwork. Table 13.3 lists the supported suboptions.
<b>type</b>	(Keyword; required if the view will be used in PDF_load_3ddata() with type=PRC) Specify the type of 3D data (default: U3D): <b>PRC</b> The view will be used in PDF_load_3ddata() with type=PRC. <b>U3D</b> The view will be used in PDF_load_3ddata() with type=U3D.
<b>U3Dpath</b>	(Hypertext string; will be ignored if the camera2world option is specified; only for type=U3D) A View Node name used to access a view node within the 3D artwork.

**Camera position.** The position of the camera can be specified with the *camera2world* option. Alternatively, JavaScript code can be attached to position and align the camera towards the model. The PDFlib Cookbook contains sample code for attaching such JavaScript code to a 3D model.

The following values can be supplied to the *camera2world* option for common camera positions. *x*, *y*, and *z* are suitable values which describe the position of the camera. These values should satisfy the stated conditions (see below):

View from the front:

{-1 0 0 0 0 1 0 1 0 x y z} *x* small, *y* large negative, *z* small

View from left:

{0 1 0 0 0 1 1 0 0 x 0 z} *x* large negative, *z* small

View from the top:

{-1 0 0 0 1 0 0 0 -1 x 0 z} *x* small, *z* large positive

View from the back:

{ 1 0 0 0 0 1 0 -1 0 x y z } x small, y large positive, z small

View from the bottom:

{ -1 0 0 0 -1 0 0 0 1 x 0 z } x small, z large negative

View from right:

{ 0 -1 0 0 0 1 -1 0 0 x 0 z } x large positive, z small

Isometric view, i.e. the direction of projection intersects all three axes at the same angle. There are exactly eight such views, one in each octant:

{ 0.707107 -0.707107 0 -0.5 -0.5 0.707107 -0.5 -0.5 -0.707107 x y z }  
x, y, z large positive

The  $x, y, z$  values should be selected depending on the position and size of the model.

»Large« means the values should be significantly larger than the size of the model in order to provide a large enough distance between the camera and the model. If the value is too large the model will appear very small and will quickly get out of sight when rotating the view. If the value is too small the model may not completely fit into the view.

»Small« means the absolute value should be small compared to the large value and should not exceed the size of the model very much.

Table 13.3 Suboptions for the rendermode option of PDF\_create\_3dview()

option	explanation
<b>crease</b>	(Float in the range 0..180) Crease value
<b>facecolor</b>	(RGB color or keyword; only for type=Illustration) Face color; this color will be used by several render modes. The keyword backgroundcolor refers to the current background color. Default: backgroundcolor
<b>opacity</b>	(Float in the range 0..1) Opacity for some render modes. Default: 0.5
<b>rendercolor</b>	(RGB color) Auxiliary color. This color will be used by several render modes. Default: black
<b>type</b>	(Option list; PDF 1.7) Specifies the render mode for displaying the 3D artwork. Supported options: (Keyword) Specifies the render mode. Supported keywords (Default: Artwork): <b>Artwork</b> Render mode is specified in the 3D artwork; all other suboptions of the rendermode option will be ignored. <b>Solid</b> Displays textured and lit geometric shapes. <b>SolidWireframe</b> Displays textured and lit geometric shapes (triangles) with single color edges on top of them. <b>Transparent</b> Displays textured and lit geometric shapes (triangles) with an added level of transparency. <b>TransparentWireframe</b> Displays textured and lit geometric shapes (triangles) with an added level of transparency. <b>BoundingBox</b> Displays textured and lit geometric shapes (triangles) with an added level of transparency, with single color opaque edges on top of it. <b>TransparentBoundingBox</b> Displays bounding boxes faces of each node, aligned with the axes of the local coordinate space for that node, with an added level of transparency. <b>TransparentBoundingBoxOutline</b> Displays bounding boxes edges and faces of each node, aligned with the axes of the local coordinate space for that node, with an added level of transparency. <b>Wireframe</b> Displays bounding boxes edges and faces of each node, aligned with the axes of the local coordinate space for that node, with an added level of transparency. <b>ShadedWireframe</b> Displays only edges, though interpolates their color between their two vertices and applies lighting. <b>HiddenWireframe</b> Displays edges in a single color, though removes back-facing and obscured edges. <b>Vertices</b> Displays only vertices in a single color. <b>ShadedVertices</b> Displays only vertices, though uses their vertex color and applies lighting. <b>Illustration</b> Displays silhouette edges with surfaces, removes obscured lines. <b>SolidOutline</b> Displays silhouette edges with lit and textured surfaces, removes obscured lines. <b>ShadedIllustration</b> Displays silhouette edges with lit and textured surfaces and an additional emissive term to remove poorly lit areas of the artwork.

## 13.2 Geospatial Features

*Note Acrobat 9 or above is required for interacting with geospatial data in PDF.*

Geospatial features are implemented with the following functions and options:

- ▶ One or more georeferenced areas can be assigned to a page with the `viewports` option of `PDF_begin/end_page_ext()`.
- ▶ The `georeference` option of `PDF_load_image()` can be used to assign an earth-based coordinate system to an image.

Table 13.4 and subsequent tables specify the options for geospatial features in detail.

Table 13.4 Suboptions for the `viewports` option of `PDF_begin/end_page_ext()`

option	explanation
<b>bounding-box</b>	(Rectangle; required) A rectangle in default coordinates specifying the location of the viewport on the page.
<b>georeference</b>	(Option list; required) Specifies the description of a world coordinate system associated with the viewport to use for geospatial measuring; see Table 13.5 for supported options.
<b>hypertext-encoding</b>	(Keyword) Specifies the encoding for the name option. An empty string is equivalent to <code>unicode</code> . Default: the value of the global <code>hypertextencoding</code> parameter
<b>name</b>	(Hypertext string) A descriptive title of the viewport (map name). However, Acrobat does not display the viewport name in the user interface.

Table 13.5 Suboptions for the `georeference` option of `PDF_load_image()` and the `georeference` suboption of the `viewports` option of `PDF_begin/end_page_ext()`

option	explanation
<b>angularunit</b>	(Keyword) Specifies the preferred angular display unit (default: <code>deg</code> ): <ul style="list-style-type: none"> <li><b>degree</b> degrees</li> <li><b>grad</b> grad (1/400 of the full circle, or 0.9 degrees)</li> </ul>
<b>areaunit</b>	(Keyword) Specifies the preferred area display unit (default: <code>sqm</code> ): <ul style="list-style-type: none"> <li><b>sqm</b> square meter</li> <li><b>ha</b> hectar (10,000 square meters)</li> <li><b>sqkm</b> square kilometer</li> <li><b>sqft</b> square foot</li> <li><b>a</b> acre</li> <li><b>sqmi</b> square mile</li> </ul> The specified unit will be used for display only if the following Acrobat setting is disabled: »Preferences, Measuring (Geo), Use Default Area Unit«.
<b>bounds</b>	(Polyline with two or more points) Specifies the bounds of an area for which the geospatial transformations are valid (for maps this bounding polyline is known as a <i>neatline</i> ). The points are expressed relative to the boundingbox of a page viewport or the extent of an image. Default: <code>{0 0 0 1 1 1 0}</code> , i.e. the full viewport or image area will be used for the map.
<b>displaysystem</b>	(Option list) Specifies a coordinate system according to Table 13.6 for the user-visible display of position values, such as latitude and longitude. This entry can be used to display the coordinates in another system than the one supplied in the <code>coords</code> option to specify the map.

Table 13.5 Suboptions for the georeference option of `PDF_load_image()` and the georeference suboption of the viewports option of `PDF_begin/end_page_ext()`

option	explanation
<b>linearunit</b>	(Keyword) Specifies the preferred linear display unit (default: m): <b>m</b> meter <b>km</b> kilometer <b>ft</b> international foot <b>usft</b> US survey foot <b>mi</b> international mile <b>nm</b> nautical mile The specified unit will be used for display only if the following Acrobat setting is disabled: »Preferences, Measuring (Geo), Use Default Distance Unit«.
<b>mappoints</b>	(List with two or more pairs of floats; required) A list of numbers where each pair defines a point in a 2D unit square. The unit square is mapped to the rectangular bounds of the page viewport or image which contains the georeference option list. The mappoints list must contain the same number of points as the worldpoints list; each point is the map position in the unit square corresponding to the geospatial position in the worldpoints list.
<b>worldpoints</b>	(List with two or more pairs of floats; required) A list of coordinate pairs where each pair specifies the world coordinates of the corresponding point in the mappoints option. The number of pairs must match the number of pairs in the mappoints option. The coordinate values are based on the coordinate system specified in the worldsystem option: if type=geographic, latitude/longitude values in degrees must be provided. If type=projected, projected x/y values must be provided.
<b>worldsystem</b>	(Option list; required) World coordinate system (for interpretation of worldpoints) according to Table 13.6.

Table 13.6 Suboptions for the mapssystem and displaysystem suboptions of the georeference option of `PDF_load_image()` and the georeference suboption of the viewports option of `PDF_begin/end_page_ext()`

option	explanation
<b>epsg</b>	(Integer; exactly one of epsg or wkt must be supplied) Specifies the coordinate system as an EPSG reference code. Note that Acrobat 9 does not support EPSG codes for type=geographic; use wkt in this case.
<b>type</b>	(Keyword; required) Specifies the type of the coordinate system: <b>geographic</b> geographic coordinate system (supports only wkt) <b>projected</b> projected coordinate system (supports wkt and epsg)
<b>wkt</b>	(String with up to 1024 ASCII characters; exactly one of epsg or wkt must be supplied) Specifies the coordinate system as a string of »Well Known Text« (WKT). WKT is recommended for custom coordinate systems without any EPSG code and seems to be required in Acrobat 9 if type=geographic.



# 14 Document Interchange

## 14.1 Document Information Fields

---

```
C++ Java void set_info(String key, String value)
Perl PHP set_info(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 14.1). 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.<sup>1</sup>

**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()*.

The document info entries will be overwritten by XMP document metadata supplied to the *metadata* option of *PDF\_begin/end\_document()*.

**Scope** *any*. If used in *object* scope the supplied values will only be used for the next document.

Table 14.1 Values for the document information field key

<b>key</b>	<b>explanation</b>
<b>Subject</b>	Subject of the document
<b>Title</b>	Title of the document. This entry must be supplied in PDF/X mode.
<b>Creator</b>	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«. This entry must be supplied in PDF/X mode.
<b>Author</b>	Author of the document
<b>Keywords</b>	Keywords describing the contents of the document
<b>Trapped</b>	Indicates whether trapping has been applied to the document. Allowed values are True, False, and Unknown. In PDF/X mode Unknown is not allowed.

1. See [dublincore.org](http://dublincore.org)

Table 14.1 Values for the document information field key

<b>key</b>	<b>explanation</b>
<b>any other name</b>	<p>User-defined document information 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 <code>moddate</code> option of <code>PDF_begin/end_document()</code>.</p> <p>Custom document info fields must not contain any of the following characters if XMP metadata is created (via the <code>autoxmp</code> or <code>metadata</code> options): <code>&amp; \ &lt; &gt; " space</code></p> <p>Fields which are used for standard identification will be rejected.</p>

## 14.2 XMP Metadata

As an alternative or in addition to document information fields PDFlib supports XMP (*Extensible Metadata Platform*<sup>1</sup>) 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 fields.** 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 fields which correspond to a well-known element in one of the standard XMP schemas will be placed in the appropriate schema. Unknown info fields will usually be placed in the extended PDF (*pdfx*) schema, but will be ignored in PDF/A mode.

**User-supplied XMP 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.

*Cookbook* A simple XMP sample can be found in the *Cookbook* topic `interchange/embed_xmp`.

For document-level metadata PDFlib will add several internally generated properties (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. Additional requirements for XMP document metadata for PDF/A are discussed in the PDFlib Tutorial.

In addition to document-level metadata, XMP can be supplied for pages, fonts, ICC profiles, images, templates, and imported PDF pages. Table 14.2 lists suboptions for the *metadata* option of various functions. Example:

```
metadata={filename=info.xmp inputencoding=winansi}
```

Table 14.2 Suboptions for the *metadata* option 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
<b>compress</b>	(Boolean; not for <i>PDF_begin/end_document()</i> ) Compress the XMP metadata stream in the PDF output. If the option is only supplied in <i>PDF_begin_page_ext()</i> but not in <i>PDF_end_page_ext()</i> , its value takes precedence over the default. Default: false PDF/A and PDF/X: <i>compress=true</i> is not allowed.
<b>inputencoding</b>	(Keyword) The encoding to interpret the metadata supplied in <i>filename</i> . Default: unicode
<b>inputformat</b>	(Keyword) The format of the metadata supplied in <i>filename</i> . Default: utf8, but bytes if <i>inputencoding</i> is an 8-bit encoding
<b>keepxmp</b>	(Boolean; only for <i>PDF_load_image()</i> ; can not be combined with <i>filename</i> ) XMP metadata present in an image will be kept, i.e. attached to the resulting image in the PDF document. XMP metadata is honored in the TIFF, JPEG, and JPEG 2000 image formats. If no XMP metadata is found in the image file this option doesn't have any effect. Default: false
<b>filename</b>	(Name string; required unless <i>keepxmp</i> is supplied) The name of a file containing well-formed XMP metadata. It will be interpreted according to the global <i>filenamehandling</i> option or parameter, see Table 2.2.

1. See [www.adobe.com/products/xmp](http://www.adobe.com/products/xmp)

## 14.3 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* document option has been set to *PDF/A-1a:2005*.

*Cookbook* A full code sample can be found in the *Cookbook* topic `interchange/starter_tagged`.

---

```
C++ Java int begin_item(String tag, String optlist)
Perl PHP int begin_item(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 14.3:

- ▶ standard element types
- ▶ pseudo element types which are not structure elements
- ▶ The tag name `Plib_custom_tag` implies use of a custom element type; the actual tag name must be supplied in the *tagname* option.

Table 14.3 Standard, pseudo, and custom types for `PDF_begin_item()`, `PDF_begin_mc()`, and `PDF_mc_point()`

category	tags	
<b>standard element types</b>		
<b>grouping</b>	Document, Part, Art, Sect, Div, BlockQuote, Caption, TOC, TOCI, Index, NonStruct, Private	
<b>paragraph-like</b>	P, H, H1-H6 (BLSEs)	
<b>list</b>	L, LI, Lbl, LBody (BLSEs)	
<b>table</b>	Table (BLSE), TR, TH, TD, THead <sup>1</sup> , TBody <sup>1</sup> , TFoot <sup>1</sup>	
<b>inline-level</b>	Span, TagSuspect <sup>2</sup> , Quote, Note, Reference, BibEntry, Code, (ILSEs)	
<b>illustration</b>	Figure, Formula, Form	
<b>Japanese</b>	Ruby <sup>1</sup> (grouping), RB <sup>1</sup> , RT <sup>1</sup> , RP <sup>1</sup> , Warichu <sup>1</sup> (grouping), WT <sup>1</sup> , WP <sup>1</sup>	
<b>pseudo element types</b>		
<b>non-structural elements</b>	<b>Artifact</b>	Specifies an artifact, to be distinguished from real page content.
	<b>ASpan</b>	(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.
	<b>ReversedChars</b>	(Not recommended) Specifies text in a right-to-left language with reversed characters.
	<b>Clip</b>	Specifies a marked clipping sequence. This is a sequence containing only clipping paths or text in rendering mode 7, but no visible graphics or <code>PDF_save()</code> / <code>PDF_restore()</code> .
<b>custom element types</b>		
<b>user-defined elements</b>	The tag name <code>Plib_custom_tag</code> must be supplied in the <i>tagname</i> parameter. The actual tag name which will be written to PDF must be supplied in the <i>tagname</i> option.	

1. Requires PDF 1.5 or above

2. Requires PDF 1.6 or above

**optlist** An option list specifying details of the item. 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:

- ▶ General option: *hypertextencoding* (see Table 12.1)
- ▶ Tag control options according to Table 14.4:  
*ActualText, Alt, artifactsubtype, artifacttype, Attached, BBox, ColSpan, E, index, inline, Lang, parent, RowSpan, Scope, tagname, 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. Pseudo element types are not allowed as parent items. 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 grouping elements also *document*; must always be paired with a matching *PDF\_end\_item()* call. This function is only allowed in Tagged PDF mode.

Table 14.4 Options for structure and pseudo tags for *PDF\_begin\_item()*, *PDF\_begin\_mc()*, and *PDF\_mc\_point()*

option	explanation
<b>ActualText</b>	(Hypertext string; not for pseudo tags except in PDF 1.5 with ASpan; not for TagSuspect) 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.
<b>Alt</b>	(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.
<b>artifact-subtype</b>	(Keyword; tag=Artifact and artifacttype=Pageination; PDF 1.7) Subtype of the artifact: Header, Footer, Watermark
<b>artifacttype</b>	(Keyword; only for tag=Artifact) Identifies the artifact type of the content item: Pageination, Layout, Page, Background (PDF 1.7)
<b>Attached</b>	(Keyword list; only for tag=Artifact and artifacttype=Pageination or artifacttype=Background with full-page background artifacts) A list containing one to four of the keywords Top, Bottom, Left, and Right
<b>BBox</b>	(Rectangle; only for tag=Artifact as well as all table and illustration tags; required for artifacttype=Background, otherwise 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.
<b>ColSpan</b>	(Integer; only for tag=TH and TD) Number of table columns spanned by a cell.
<b>E</b>	(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.
<b>index</b>	(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

Table 14.4 Options for structure and pseudo tags for `PDF_begin_item()`, `PDF_begin_mc()`, and `PDF_mc_point()`

option	explanation
<b>inline</b>	(Boolean; only for <code>tag=ASpan</code> and all inline-level tags except <code>TagSuspect</code> ) If <code>true</code> , the content item will be written inline, and no structure element will be created. Default: <code>true</code>
<b>Lang</b>	(String; not for <code>TagSuspect</code> and pseudo tags except <code>ASpan</code> ) Language identifier for the content item in the format described in Table 3.1 for the <code>lang</code> option. This can be used to override the document's dominant language for individual content items.
<b>parent</b>	(Item handle; not for <code>TagSuspect</code> and pseudo tags) The item handle of the element's parent, as returned by another call to <code>PDF_begin_item()</code> . The value 0 refers to the structure tree root. -1 refers to the currently active element. In other words, <code>parent=-1</code> opens a child of the current element. Pseudo element types are not allowed as parent items. Default: -1
<b>RowSpan</b>	(Integer; only for <code>tag=TH</code> and <code>TD</code> ) The number of table rows spanned by a cell.
<b>Scope</b>	(Keyword; only for <code>tag=TH</code> ; PDF 1.5 or above) One of the keywords <code>Row</code> , <code>Column</code> , or <code>Both</code> 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.
<b>tagname</b>	(Name string; only for <code>tag=Plib_custom_tag</code> , and required in this case) Arbitrary name of a custom tag for which a mapping to a standard element type must have been defined with the <code>rolemap</code> option of <code>PDF_begin_document()</code> . Custom element types are restricted to 127 winansi characters or a sequence containing arbitrary Unicode characters provided the length of the corresponding UTF-8 sequence does not exceed 127 bytes.
<b>Title</b>	(Hypertext string; not for inline and pseudo tags) Name of the structure element

---

```

C++ Java void end_item(int id)
Perl PHP end_item(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 activate_item(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.

## 14.4 Marked Content

---

**C++ Java** `void begin_mc(String tag, String optlist)`  
**Perl PHP** `begin_mc(string tag, string optlist)`  
**C** `void PDF_begin_mc(PDF *p, const char *tag, const char *optlist)`

---

Begin a marked content sequence with optional properties.

**tag** The name of the marked content sequence. The following tags are supported:

- ▶ All inline-level and pseudo tags in Table 14.3.
- ▶ The tag `Plib_custom` can be used for custom entries with user-defined properties.
- ▶ The tag `Plib` is reserved.

**optlist** The following options for marked content sequences are supported:

- ▶ Options for standard properties of the according to Table 14.4.
- ▶ The tags `Plib_custom` and `Plib` additionally support the option in Table 14.5.

**Details** A marked content sequence with the specified tag and properties will be started. If no options are provided a sequence without any properties will be created. Marked content sequences can be nested to an arbitrary level. The user is responsible for creating properly nested sequences of `PDF_begin/end_item()` and `PDF_begin/end_mc()`.

**Scope** `page`, `pattern`, `template`, `glyph`, must always be paired with a matching `PDF_end_mc()` call in the same `page`, `pattern`, `template`, or `glyph` scope.

Table 14.5 Option for user-defined properties of tags with `PDF_begin_mc()` and `PDF_mc_point()`

option	explanation
<b>properties</b>	(List of option lists; only for tag= <code>Plib</code> and tag= <code>Plib_custom</code> ) Each list contains three options which specify a user-defined property:
<b>key</b>	(String; required) Name of the property.
<b>type</b>	(Keyword; required) Type of the property value: boolean, name, or string.
<b>value</b>	(Hypertext string if type= <code>string</code> , otherwise string; required) Value of the property.

---

**C++ Java** `void end_mc()`  
**Perl PHP** `end_mc()`  
**C** `void PDF_end_mc(PDF *p)`

---

End the least recently opened marked content sequence.

**Details** All marked content sequences must be closed before calling `PDF_end_page_ext()`.

**Scope** `page`, `pattern`, `template`, `glyph`, must always be paired with a matching `PDF_begin_mc()` call in the same `page`, `pattern`, `template`, or `glyph` scope.

---

**C++ Java** `void mc_point(String tag, String optlist)`

**Perl PHP** `mc_point(string tag, string optlist)`

**C** `void PDF_mc_point(PDF *p, const char *tag, const char *optlist)`

---

Add a marked content point with optional properties.

**tag** The name of the marked content point. The following tags are supported:

- ▶ All inline-level and pseudo tags in Table 14.3.
- ▶ The tag *Plib\_custom* can be used for custom entries.
- ▶ The tag *Plib* is reserved.

**optlist** The following options are supported:

- ▶ Options for standard properties of the marked content point according to Table 14.4.
- ▶ The tags *Plib\_custom* and *Plib* additionally support the option in Table 14.5.

**Details** A marked content point with the specified tag and properties will be created. If no options are provided a marked content point without any properties will be created.

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



# A List of all Functions

This appendix lists all API functions. Click on a function name to jump to the corresponding description.

<b>General</b>	<b>Font</b>	<b>Text Formatting</b>	<b>Color</b>
<a href="#">get_value</a>	<a href="#">load_font</a>	<a href="#">fit_textline</a>	<a href="#">setcolor</a>
<a href="#">set_value</a>	<a href="#">close_font</a>	<a href="#">info_textline</a>	<a href="#">makespotcolor</a>
<a href="#">get_parameter</a>	<a href="#">setfont</a>	<a href="#">add_textflow</a>	<a href="#">load_iccprofile</a>
<a href="#">set_parameter</a>	<a href="#">info_font</a>	<a href="#">create_textflow</a>	<a href="#">begin_pattern</a>
<a href="#">set_option</a>	<a href="#">begin_font</a>	<a href="#">fit_textflow</a>	<a href="#">end_pattern</a>
<a href="#">new</a>	<a href="#">end_font</a>	<a href="#">info_textflow</a>	<a href="#">shading_pattern</a>
<a href="#">delete</a>	<a href="#">begin_glyph</a>	<a href="#">delete_textflow</a>	<a href="#">shfill</a>
<a href="#">begin_document</a>	<a href="#">end_glyph</a>		<a href="#">shading</a>
<a href="#">begin_document_callback</a>	<a href="#">encoding_set_char</a>	<b>Table Formatting</b>	
<a href="#">end_document</a>		<a href="#">add_table_cell</a>	<b>Image</b>
<a href="#">get_buffer</a>	<b>Text Output</b>	<a href="#">fit_table</a>	<a href="#">load_image</a>
	<a href="#">set_text_pos</a>	<a href="#">info_table</a>	<a href="#">close_image</a>
<a href="#">begin_page_ext</a>	<a href="#">show</a>	<a href="#">delete_table</a>	<a href="#">fit_image</a>
<a href="#">end_page_ext</a>	<a href="#">xshow</a>		<a href="#">info_image</a>
<a href="#">suspend_page</a>	<a href="#">show_xy</a>	<b>Matchboxes</b>	<a href="#">begin_template_ext</a>
<a href="#">resume_page</a>	<a href="#">continue_text</a>	<a href="#">info_matchbox</a>	<a href="#">end_template_ext</a>
<a href="#">define_layer</a>	<a href="#">stringwidth</a>		<a href="#">add_thumbnail</a>
<a href="#">set_layer_dependency</a>			
<a href="#">begin_layer</a>	<b>Unicode Conversion</b>		
<a href="#">end_layer</a>	<a href="#">utf16_to_utf8</a>		
	<a href="#">utf8_to_utf16</a>		
<a href="#">create_pvf</a>	<a href="#">utf32_to_utf16</a>		
<a href="#">delete_pvf</a>	<a href="#">utf8_to_utf32</a>		
<a href="#">get_errnum</a>	<a href="#">utf32_to_utf8</a>		
<a href="#">get_errmsg</a>	<a href="#">utf16_to_utf32</a>		
<a href="#">get_apiname</a>			
<a href="#">get_opaque</a>			

**Graphics State**

*setdash*  
*setdashpattern*  
*setflat*  
*setlinejoin*  
*setlinecap*  
*setmiterlimit*  
*setlinewidth*  
*initgraphics*  
*save*  
*restore*  
*create\_gstate*  
*set\_gstate*

**Coordinate Transformation**

*translate*  
*scale*  
*rotate*  
*align*  
*skew*  
*concat*  
*setmatrix*

**Path Construction**

*moveto*  
*lineto*  
*curveto*  
*circle*  
*arc*  
*arcn*  
*circular\_arc*  
*ellipse*  
*rect*  
*closepath*

**Path Painting and Clipping**

*stroke*  
*closepath\_stroke*  
*fill*  
*fill\_stroke*  
*closepath\_fill\_stroke*  
*clip*  
*endpath*

**Path Objects**

*add\_path\_point*  
*draw\_path*  
*info\_path*  
*delete\_path*

**PDI**

*open\_pdi\_document*  
*open\_pdi\_callback*  
*close\_pdi\_document*  
*open\_pdi\_page*  
*close\_pdi\_page*  
*fit\_pdi\_page*  
*info\_pdi\_page*  
*process\_pdi*

**PCOS**

*pcos\_get\_number*  
*pcos\_get\_string*  
*pcos\_get\_stream*

**Block Filling (PPS)**

*fill\_textblock*  
*fill\_imageblock*  
*fill\_pdfblock*

**Interactive Features**

*create\_action*  
*add\_nameddest*  
*create\_annotation*  
*create\_field*  
*create\_fieldgroup*  
*create\_bookmark*  
*add\_portfolio\_folder*  
*add\_portfolio\_file*

**Multimedia**

*load\_3ddata*  
*create\_3dview*

**Document Interchange**

*set\_info*  
*begin\_item*  
*end\_item*  
*activate\_item*  
*begin\_mc*  
*end\_mc*  
*mc\_point*

# B List of all Parameters

This appendix lists all keywords for *PDF\_get/set\_parameter()* and *PDF\_get/set\_value()*. Click on a keyword to jump to the corresponding description.

<b>setup</b>	<b>logging</b>	<b>font handling</b>	<b>simple text output</b>
<i>PDF_get/set_parameter()</i> :	<i>PDF_get/set_parameter()</i> :	<i>PDF_get/set_parameter()</i> :	<i>PDF_get/set_parameter()</i> :
<i>errorpolicy</i>	<i>logging</i> <sup>1</sup>	<i>Encoding</i>	<i>autospace</i>
<i>filenamehandling</i>	<i>userlog</i> <sup>1</sup>	<i>FontAFM</i>	<i>charref</i>
<i>resourcefile</i>		<i>FontPFM</i>	<i>decorationabove</i>
<i>scope</i>	<b>versioning</b>	<i>FontOutline</i>	<i>escapesequence</i>
<i>SearchPath</i> <sup>1</sup>	<i>PDF_get/set_parameter()</i> :	<i>HostFont</i>	<i>fakebold</i>
<i>string</i> <sup>1</sup>	<i>version</i> <sup>21</sup>	<i>PDF_get/set_value()</i> :	<i>glyphcheck</i>
<i>asciifile</i>	<i>PDF_get/set_value()</i> :	<i>font</i> <sup>2</sup>	<i> Kerning</i>
	<i>major</i>	<i>fontsize</i> <sup>2</sup>	<i>underline, overline, strikeout</i>
<i>PDF_get/set_value()</i> :	<i>minor</i>		<i>textformat</i>
<i>compress</i>	<i>revision</i> <sup>2</sup>	<b>image</b>	<i>PDF_get/set_value()</i> :
<i>maxfilehandles</i>		<i>PDF_get/set_parameter()</i> :	<i>charspacing</i>
	<b>page</b>	<i>honoriccprofile</i>	<i>horizscaling</i>
<b>graphics</b>	<i>PDF_get/set_parameter()</i> :	<i>renderingintent</i>	<i>italicangle</i>
<i>PDF_get/set_parameter()</i> :	<i>topdown</i>		<i>leading</i>
<i>fillrule</i>	<i>PDF_get/set_value()</i> :		<i>strokewidth</i>
<i>PDF_get/set_value()</i> :	<i>pagewidth</i>		<i>textrendering</i>
<i>currentx</i> <sup>2</sup> , <i>currenty</i> <sup>2</sup>	<i>pageheight</i>		<i>textrise</i>
<i>ctm_a</i> <sup>2</sup> , <i>ctm_b</i> <sup>2</sup> , <i>ctm_c</i> <sup>2</sup>			<i>textx</i> <sup>2</sup> , <i>texty</i> <sup>2</sup>
<i>ctm_d</i> <sup>2</sup> , <i>ctm_e</i> <sup>2</sup> , <i>ctm_f</i> <sup>2</sup>	<b>ICC profiles</b>		<i>underlineposition</i>
	<i>PDF_get/set_parameter()</i> :		<i>underlinewidth</i>
<b>color</b>	<i>ICCProfile</i>	<b>interactive</b>	<i>wordspacing</i>
<i>PDF_get/set_parameter()</i> :	<i>StandardOutputIntent</i>	<i>PDF_get/set_parameter()</i> :	
<i>preserveoldpantone names</i>	<i>PDF_get/set_value()</i> :	<i>usercoordinates</i>	<b>PDI</b>
<i>spotcolorlookup</i>	<i>icccomponents</i> <sup>2</sup>	<i>hypertextencoding</i>	<i>PDF_get/set_parameter()</i> :
	<i>setcolor:iccprofilegray</i>	<i>hypertextformat</i>	<i>pdi</i> <sup>1</sup>
	<i>setcolor:iccprofilergb</i>	<i>usehypertextencoding</i>	
	<i>setcolor:iccprofilecmyk</i>		

1. Only for *PDF\_get/set\_parameter()*

2. Only for *PDF\_get/set\_value()*



# C List of all Options and Keywords

This index contains an alphabetical list of all options and keywords along with the functions in which they can be used. Click on the page number to jump to the description.

## &

**&name** option list macro call in `fit_textflow()` 89

## 3D

**3Dactivate** in `create_annotation()` 195

**3Ddata** in `create_annotation()` 195

**3Dinitialview** in `create_annotation()` 195

**3Dinteractive** in `create_annotation()` 195

**3Dshared** in `create_annotation()` 195

**3Dview** in `create_action()` 188

## A

**acrobat** suboption for fontname in `info_font()` 60  
**action**

in `begin/end_page_ext()` 41

in `create_annotation()` 196

in `create_bookmark()` 209

in `create_field()` and `create_fieldgroup()` 204

in `end_document()` 32

in `process_pdi()` 176

**actual** suboption for encoding in `info_font()` 59

**ActualText** in `begin_item()` 229

**addfitbox** suboption for wrap in `fit_textflow()` 96

**adjustmethod** in `add/create_textflow()` 86

**adjustpage**

in `fit_image/pdi_page()` 160

in `fit_pdi_page()` 173

**advancedlinebreak** in `add/create_textflow()` 86

**align** in `draw_path()` 111

**alignchar** in `fit/info_textline()` 111

**alignment**

in `add/create_textflow()` 84

in `create_annotation()` 196

suboption for leader in `fit/info_textline()` and `add/create_textflow()` 78

**alphachannelname** in `load_image()` 156

**alphaishape** in `create_gstate()` 125

**Alt** in `begin_item()` 229

**angle** keyword in `info_textline()` 82

**angularunit** suboption for georeference 222

**annotation** suboption for targetpath in `create_action()` 191

**annotationtype** in `add_table_cell()` 100

**annotacolor** in `create_annotation()` 196

**antialias**

in `shading()` 151

suboption for shading option of several functions 121

**api**

suboption for encoding in `info_font()` 59

suboption for fontname in `info_font()` 60

**area** suboption for fill in `fit_table()` 104

**areaunit** suboption for georeference 222

**artbox** in `begin/end_page_ext()` 41

**artifactssubtype** in `begin_item()` 229

**artfacttype** in `begin_item()` 229

**ascender**

in `info_font()` 59

in `load_font()` 53

keyword in `info_textline()` 82

**Attached** in `begin_item()` 229

**attachmentpassword** in `begin_document()` 32

**attachmentpoint** in `draw_path()` 111

**attachments** in `begin/end_document()` 33

**autocidfont** in `load_font()` 53

**autosubsetting** in `load_font()` 53

**autoxmp** in `begin/end_document()` 33

**avoidbreak** in `add/create_textflow()` 86

**avoiddemostamp** in `set_option()` 20

**avoidemptybegin** in `add/create_textflow()` 84

## B

**background** in `create_3dview()` 218

**backgroundcolor** in `create_field()` and `create_fieldgroup()` 204

**barcode** in `create_field()` and `create_fieldgroup()` 204

**basestate** in `set_layer_dependency()` 47

**BBox** in `begin_item()` 229

**begoptlistchar** in `create_textflow()` 90

**beziers** suboption for wrap in `fit_textflow()` 96

**bitreverse** in `load_image()` 156

**bleedbox** in `begin/end_page_ext()` 41

**blendmode** in `create_gstate()` 125

**blind**

in `fit_table()` 103

in `fit_textflow()` 92

in many functions 111

**bordercolor** in `create_field()` and `create_fieldgroup()` 204

**borderstyle**

in `create_annotation()` 196

in `create_field()` and `create_fieldgroup()` 204

**borderwidth** in several functions 119

**bottom** in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 192

**boundingbox**

- in `shading()` 151
- keyword in `info_image()` 161
- keyword in `info_path()` 139
- keyword in `info_pdi_page()` 174
- keyword in `info_table()` 106
- keyword in `info_textflow()` 97
- suboption for `viewports` option in `begin/end_page_ext()` 222

**bounds** suboption for `georeference` 222**boxes** suboption for `wrap` in `fit_textflow()` 96**boxexpand** in `open_pdi_page()` 172**boxheight** suboption for `matchbox` 115**boxlinecount** keyword in `info_textflow()` 97**boxsize** in various functions 111**boxwidth** suboption for `matchbox` 115**bpc** in `load_image()` 156**buttonlayout** in `create_field()` and `create_fieldgroup()` 204**buttonstyle** in `create_field()` and `create_fieldgroup()` 204**C****calcorder** in `create_field()` and `create_fieldgroup()` 204**calloutline** in `create_annotation()` 196**camerazworld** in `create_3dview()` 218**cameradistance** in `create_3dview()` 219**canonicaldate** in `create_action()` 188**capheight**

- in `info_font()` 59
- in `load_font()` 53
- keyword in `info_textline()` 82

**caption**

- in `create_field()` and `create_fieldgroup()` 205
- suboption for the `barcode` option in `create_field()` and `create_fieldgroup()` 208

**captiondown** in `create_field()` and `create_fieldgroup()` 205**captionoffset** in `create_annotation()` 196**captionposition** in `create_annotation()` 196**captionrollover** in `create_field()` and `create_fieldgroup()` 205**cascadedflate** in `load_image()` 156**centerwindow** suboption for `viewerpreferences`

- in `begin/end_document()` 37

**charclass** in `add/create_textflow()` 87**charmapping** in `add/create_textflow()` 88**charref** in many functions 75**charspacing**

- in `create_field()` and `create_fieldgroup()` 205
- in many functions 75

**checkwordsplitting** in `add_table_cell()` 100**children** in `set_layer_dependency()` 47**cid** in `info_font()` 58, 59**cidfont** in `info_font()` 59**circles** suboption for `wrap` in `fit_textflow()` 96**circular** keyword in `add_path_point()` 136**classes** for logging parameter 30**clip** in `draw_path()` 138**clipping** suboption for `matchbox` 115**clippingarea** in `open_pdi_page()` 172**clippingpath** keyword in `info_image()` 161**clippingpathname** in `load_image()` 156**cloneboxes**

- in `fit_pdi_page()` 174
- in `open_pdi_page()` 172

**close**

- in `add_path_point()` 137
- in `draw_path()` 138

suboption for `textpath` in `fit_textline()` 81**cloudy** in `create_annotation()` 196**code** in `info_font()` 58, 59**codepage** in `info_font()` 59**codepagelist** in `info_font()` 59**colorize** in `load_image()` 156**colored** in `begin_font()` 62**colscalegroup** in `add_table_cell()` 100**colspan** in `add_table_cell()` 100**ColSpan** in `begin_item()` 229**colwidth** in `add_table_cell()` 100, 101**colwidthdefault** in `fit_table()` 103**comb** in `create_field()` and `create_fieldgroup()` 205**comment** option list macro definition in `fit_textflow()` 87**commitonselect** in `create_field()` and `create_fieldgroup()` 205**compatibility** in `begin_document()` 33**components** in `load_image()` 156**compress** suboption for `metadata` 227**contents** in `create_annotation()` 196**continuetextflow** in `add_table_cell()` 100**control** keyword in `add_path_point()` 136**convert** in `pcos_get_stream()` 179**copy** in `create_pvf()` 25**copyglobals** in `load_image()` 156**coversheet** suboption for `portfolio` in `begin_document()` 214**coversheetfolder** suboption for `portfolio` in `begin_document()` 214**crease** suboption for `rendermode` in `create_3dview()` 221**createdate** in `create_annotation()` 196**createfittext** in `fit_textflow()` 92**createlastindent** in `fit_textflow()` 92**creatematchboxes** suboption for `wrap` in `fit_textflow()` 96**createorderlist** in `set_layer_dependency()` 48**createpvf** in `begin_document()` 33**createrichtext** in `create_annotation()` 197**createwrapbox** suboption for `matchbox` 115**creatorinfo** in `define_layer()` 45**cropbox** in `begin/end_page_ext()` 41**currentvalue** in `create_field()` and `create_fieldgroup()` 205

*curve* keyword in `add_path_point()` 136  
*custom* in `create_annotation()` 197

## D

### **dasharray**

in `add_path_point()` 136  
in `create_annotation()` 197  
in `create_field()` and `create_fieldgroup()` 205  
in many functions 75  
in `setdashpattern` 122  
in several functions 119

### **dashphase**

in `add_path_point()` 136  
in `setdashpattern` 122  
in several functions 119

### **dataprep**

suboption for the barcode option in  
`create_field()` and `create_fieldgroup()` 208

**debugshow** in `fit_table()` 103

### **decorationabove**

in `fit/info_textline()` and `add/  
create_textflow()` 66  
in many functions 75

**defaultcmyk** in `begin_page_ext()` 41

**defaultdir** in `create_action()` 188

**defaultgray** in `begin_page_ext()` 41

**defaultrgb** in `begin_page_ext()` 41

**defaultstate** in `define_layer()` 45

**defaultvalue** in `create_field()` and  
`create_fieldgroup()` 205

**defaultvariant** in `set_layer_dependency()` 48

**defaultview** in `load_3d()` 217

**depend** in `set_layer_dependency()` 48

### **descender**

in `info_font()` 59  
in `load_font()` 53  
keyword in `info_textline()` 82

### **description**

in `add_portfolio_file()` 212  
in `add_portfolio_folder()` 211  
in `load_iccprofile()` 145  
suboption for attachments in `begin/  
end_document()` 33

### **destination**

in `begin/end_document()` 33  
in `create_action()` 188  
in `create_annotation()` 197  
in `create_bookmark()` 209

### **destname**

in `create_action()` 189  
in `create_annotation()` 197  
in `create_bookmark()` 209  
in `end_document()` 33  
suboption for `targetpath` in `create_action()`  
191

**direction** suboption for viewerpreferences in  
`begin/end_document()` 37

### **disable**

for logging parameter 29  
suboption for `3Dactivate` in  
`create_annotation()` 201

**disablestate** suboption for `3Dactivate` in  
`create_annotation()` 201

### **display**

in `create_annotation()` 197  
in `create_field()` and `create_fieldgroup()` 205

**displaydoctitle** suboption for viewerpreferences  
in `begin/end_document()` 37

**displaysystem** suboption for georeference 222

### **domain**

in `shading()` 151  
suboption for shading option of several  
functions 121

**doubleadapt** suboption for matchbox 115

**doubleoffset** suboption for matchbox 115

### **down**

suboption for template in  
`create_annotation()` 200

**dpi** in many functions 111

**drawbottom, drawleft, drawright, drawtop**

suboptions for matchbox 115

**dropcorewidths** in `load_font()` 53

**duplex** suboption for viewerpreferences in `begin/  
end_document()` 37

### **duration**

in `begin/end_page_ext()` 41  
in `create_action()` 189

## E

**E** in `begin_item()` 229

### **ecc**

suboption for the barcode option in  
`create_field()` and `create_fieldgroup()` 208

**editable** in `create_field()` and `create_fieldgroup()`  
205

**embedding** in `load_font()` 53

**embedprofile** in `load_iccprofile()` 145

### **enable**

for logging parameter 29  
suboption for `3Dactivate` in  
`create_annotation()` 201

**enablestate** suboption for `3Dactivate` in  
`create_annotation()` 201

### **encoding**

in `info_font()` 59  
in `load_font()` 54

### **end**

suboption for matchbox 116  
suboption for shading option of several  
functions 121

**endcolor** suboption for shading option of several  
functions 121

**endingstyles** in `create_annotation()` 197

**endoptlistchar** in `create_textflow()` 90

**endx, endy** keywords in `info_textline()` 82

**entire** suboption for background in `create_3dview()` 218

**eps** suboption for the coords and displaycoords suboptions of `georeference` 223

**errpolicy** parameter and option for various functions 27

**escapesequence** in many functions 75

**exceedlimit** suboption for `matchbox` 116

**exchangeFillColor** in `fit_textflow()` 92

**exchangeStrokeColors** in `fit_textflow()` 92

**exclude** in `create_action()` 189

**exists** keyword in `info_matchbox()` 117

**exportable** in `create_field()` and `create_fieldgroup()` 205

**exportmethod** in `create_action()` 189

**extendo, extend1** in `shading()` 151

## F

**facecolor** suboption for rendermode in `create_3dview()` 221

**fakebold** in many functions 75

**faked**  
suboption for ascender in `info_font()` 59  
suboption for fontstyle in `info_font()` 60

**fallbackfont** in `info_font()` 59

**fallbackfonts** in `load_font()` 54

**familyname**  
in `begin_font()` 62  
in `info_font()` 59

**feature** in `info_font()` 60

**featurelist** in `info_font()` 60

**features** in many functions 77

**fieldlist**  
in `add_portfolio_file()` 212  
in `add_portfolio_folder()` 211

**fieldname** in `add_table_cell()` 100

**fieldtype**  
in `add_table_cell()` 100  
in `create_fieldgroup()` 205

**filemode** in `begin_document()` 33

**filename**  
for logging parameter 29  
in `create_action()` 189  
in `create_annotation()` 197  
suboption for attachments in `begin/end_document()` 33  
suboption for metadata 227  
suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 163  
suboption for search in `begin/end_document()` 36

**filename** keyword in `info_image()` 161

**filenamehandling** in `set_option()` 21

**fileselect** in `create_field()` and `create_fieldgroup()` 205

## fill

in `draw_path()` 137, 138  
in `fit_table()` 104

## fillcolor

in `add_path_point()` 136  
in `create_annotation()` 197  
in `create_field()` and `create_fieldgroup()` 205  
in many functions 76  
in several functions 119  
suboption for background in `create_3dview()` 218  
suboption for leader in `fit/info_textline()` and `add/create_textflow()` 78  
suboption for leader in `fit_textline()` 80

## fillrule

in `add_path_point()` 136  
in several functions 119  
suboption for wrap in `fit_textflow()` 96

## firstbodyrow

keyword in `info_matchbox()` 117  
keyword in `info_table()` 106

## firstdraw

in `fit_table()` 104

## firstlinedist

in `fit_textflow()` 93  
keyword in `info_textflow()` 97

## firstparalinecount

keyword in `info_textflow()` 97

## fitannotation

in `add_table_cell()` 100

## fitfield

in `add_table_cell()` 100

## fitimage

in `add_table_cell()` 100

## fitmethod

in `create_field()` and `create_fieldgroup()` 206  
in `fit_textflow()` 93

## fitpath

in various functions 111

## fitpdipage

suboption for template in `create_annotation()` 200

## fitscalex, fitscaley

keywords in `info_image()` 161  
keywords in `info_pdi_page()` 174

## fittext

keyword in `info_textflow()` 97

## fittextflow

in `add_table_cell()` 101

## fittextline

in `add_table_cell()` 101

## fitwindow

suboption for viewerpreferences in `begin/end_document()` 37

## fixedleading

in `add/create_textflow()` 85

## fixedtextformat

in `create_textflow()` 90

## flatness

in `add_path_point()` 136  
in `create_gstate()` 125  
in several functions 119

## flush

for logging parameter 29  
in `begin_document()` 33

## font

- in `create_annotation()` 198
- in `create_field()` and `create_fieldgroup()` 206
- in many functions 76
- suboption for leader in `fit/info_textline()` and `add/create_textflow()` 78

**fontfile** in `info_font()` 60

## fontname

- in `info_font()` 60
- in `load_font()` 54

## fontscale

- in `fit_textflow()` 93
- keyword in `info_textflow()` 97

## fontsize

- in `create_annotation()` 198
- in `create_field()` and `create_fieldgroup()` 206
- in many functions 76
- suboption for ascender in `info_font()` 59
- suboption for leader in `fit/info_textline()` and `add/create_textflow()` 78

## fontstyle

- in `create_bookmark()` 209
- in `info_font()` 60
- in `load_font()` 54

**fonttype** in `info_font()` 60

**footer** in `fit_table()` 104

**forcebox** in `open_pdi_page()` 172

**full** suboption for **fontname** in `info_font()` 60

## G

### georeference

- in `begin_template_ext()` 162
- in `load_image()` 156
- suboption for viewports in `begin/end_page_ext()` 222

**glyphcheck** in many functions 75

**glyphid** in `info_font()` 58, 60

**glyphname** in `info_font()` 58, 60

### group

- in `begin_page_ext()` 41
- in `resume_page()` 44
- in `set_layer_dependency()` 48
- option in `add_nameddest()` and suboption for destination option in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 192
- suboption for labels in `begin_document()` 36

**groups** in `begin_document()` 33

## gstate

- in `add_path_point()` 136
- in `fit/info_textline()` and `add/create_textflow()` 93
- in `fit_image/pdi_page()` 160
- in `fit_pdi_page()` 173
- in `fit_table()` 104
- in many graphics functions 119
- in many text functions 76
- in `shading_pattern()` 150
- suboption for shadow in `fit_textline()` 80

## H

**header** in `fit_table()` 104

### height

- in `begin/end_page_ext()` 41
- in `load_image()` 156
- keyword in `info_image()` 161
- keyword in `info_matchbox()` 117
- keyword in `info_path()` 139
- keyword in `info_pdi_page()` 174
- keyword in `info_table()` 106
- keyword in `info_textline()` 82

**hide** in `create_action()` 189

**hidemenubar** suboption for viewerpreferences in `begin/end_document()` 38

**hidetoolbar** suboption for viewerpreferences in `begin/end_document()` 38

**hidewindowui** suboption for viewerpreferences in `begin/end_document()` 38

### highlight

- in `create_annotation()` 198
- in `create_field()` and `create_fieldgroup()` 206

**honorclippingpath** in `load_image()` 156

**honoriccprofile** in `load_image()` 156

**horboxgap** keyword in `info_table()` 106

**horizscaling** in many functions 76

**horshrinking** keyword in `info_table()` 106

**horshrinklimit** in `fit_table()` 104

**hortabmethod** in `add/create_textflow()` 85

**hortabsize** in `add/create_textflow()` 85

**hostfont** in `info_font()` 60

### hypertextencoding

- parameter and option for various functions 187
- suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 36
- suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 163
- suboption for viewports in `begin/end_page_ext()` 222

**hypertextformat** parameter and option for various functions 187

**hyphenchar** in `add/create_textflow()` 88

## I

**iccprofile**  
in `load_image()` 156  
keyword in `info_image()` 161

**icon** in `create_field()` and `create_fieldgroup()` 206

**icondown** in `create_field()` and `create_fieldgroup()` 206

**iconname**  
in `begin_template_ext()` 162  
in `create_annotation()` 198  
in `load_image()` and `begin_template_ext()` 156  
in `open_pdi_page()` 172

**iconrollover** in `create_field()` and `create_fieldgroup()` 206

**ignoreclippingpath** in `fit_image/pdi_page()` 160

**ignoremask** in `load_image()` 156

**ignoreorientation**  
in `fit_image/pdi_page()` 160  
in `load_image()` 157

**image** in `add_table_cell()` 101

**imagehandle** in `load_image()` 157

**imageheight** keyword in `info_image()` 161

**imagemask** keyword in `info_image()` 161

**imagetype** keyword in `info_image()` 161

**imagewidth** keyword in `info_image()` 161

**index**  
in `begin_item()` 229  
in `create_bookmark()` 209

**indextype** suboption for search in `begin/end_document()` 36

**infomode** in `open_pdi_document()` 168

**initialexportstate** in `define_layer()` 45

**initialprintstate** in `define_layer()` 45

**initialsubset** in `load_font()` 55

**initialview** suboption for portfolio in `begin_document()` 214

**initialviewstate** in `define_layer()` 45

**inittextstate** in `fit/info_textline()` 80

**inline**  
in `begin_item()` 230  
in `load_image()` 157

**inmemory**  
in `begin_document()` 34  
in `open_pdi_document` 168

**innerbox** suboption for `matchbox` 116

**inputencoding** suboption for metadata 227

**inputformat** suboption for metadata 227

**inreplyto** in `create_annotation()` 198

**intent** in `define_layer()` 45

**interiorcolor** in `create_annotation()` 198

**interpolate** in `load_image()` 157

**inversefill** suboption for `wrap` in `fit_textflow()` 96

**invert** in `load_image()` 157

**invisiblelayers** in `set_layer_dependency()` 48

**ismap** in `create_action()` 189

## italicangle

in `info_font()` 60  
in many functions 76

**itemname** in `create_field()` and `create_fieldgroup()` 206

**itemnamelist** in `create_field()` and `create_fieldgroup()` 206

**itemtextlist** in `create_field()` and `create_fieldgroup()` 206

## K

**K** in `load_image()` 157

**keepfilter** in `pcos_get_stream()` 179

**keepfont** in `load_font()` 55

**keephandles** in `delete_table()` 107

**keepnative**  
in `info_font()` 60  
in `load_font()` 55

**keepxmp** suboption for metadata 227

**Kerning** in many functions 76

**kerningpairs** in `info_font()` 60

**key**  
suboption for custom in `create_annotation()` 197  
suboption for fieldlist in `add_portfolio_folder()` and `add_portfolio_file()` 213  
suboption for properties in `begin_mc()` and `mc_point()` 232

## L

**label** in `begin/end_page_ext()` 41

**labels** in `begin/end_document()` 34

**lang** in `begin_document()` 34

**Lang** in `begin_item()` 230

**language**  
in `define_layer()` 46  
in many functions 77  
suboption for feature in `info_font()` 60

**lastalignment** in `add/create_textflow()` 85

**lastbodyrow** keyword in `info_table()` 106

**lastfont** keyword in `info_textflow()` 97

**lastfontsize** keyword in `info_textflow()` 97

**lastlinedist**  
in `fit_textflow()` 93  
keyword in `info_textflow()` 97

**lastmark** keyword in `info_textflow()` 97

**lastparalinecount** keyword in `info_textflow()` 97

**layer**  
in `begin_template_ext()` 162  
in `create_annotation()` 198  
in `create_field()` and `create_fieldgroup()` 206  
in `load_image()` and `begin_template_ext()` 157  
in `open_pdi_page()` 172

**layerstate** in `create_action()` 189

## **leader**

in `add/create_textflow()` 85  
in `fit/info_textline()` 80

**leaderlength** in `create_annotation()` 198

## **leaderoffset**

in `create_annotation()` 198

## **leading**

in `add/create_textflow()` 85  
keyword in `info_textflow()` 97

**left** option in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 192

**leftindent** in `add/create_textflow()` 85

**leftlinex**, **leftliney** keywords in `info_textflow()` 97

**license** in `set_option()` 21

**licensefile** in `set_option()` 21

**lighting** in `create_3dview()` 219

## **line**

in `create_annotation()` 198  
keyword in `add_path_point()` 136  
suboption for `stroke` in `fit_table()` 105

**linearize** in `begin_document()` 34

**linearunit** suboption for `georeference` 223

## **linecap**

in `add_path_point()` 136  
in `create_gstate()` 125  
in `load_font()` 55  
in several functions 120

**linegap** in `info_font()` 60

**lineheight** suboption for `wrap` in `fit_textflow()` 96

## **linejoin**

in `add_path_point()` 136  
in `create_gstate()` 125  
in several functions 120

**linespreadlimit** in `fit_textflow()` 93

## **linewidth**

in `add_path_point()` 136  
in `create_annotation()` 198  
in `create_field()` and `create_fieldgroup()` 206  
in `create_gstate()` 125  
in several functions 120

**listmode** in `set_layer_dependency()` 48

**locale** in `add/create_textflow()` 86

## **locked**

in `create_annotation()` 199  
in `create_field()` and `create_fieldgroup()` 206

**lockedcontents** in `create_annotation()` 199

**lockmode** in `create_field()` and `create_fieldgroup()` 206

**logging** in `set_option()` 21

# **M**

**macro** option list macro definition in `fit_textflow()` 89

**maingid** in `info_font()` 60

**mappoints** suboption for `georeference` 223

**mapsystem** suboption for `georeference` 223

## **margin**

in `add_table_cell()` 101  
in various functions 112  
suboption for `matchbox` 116

**marginbottom** in `add_table_cell()` 101

**marginleft** in `add_table_cell()` 101

**marginright** in `add_table_cell()` 101

**mintop** in `add_table_cell()` 101

**mark** in `add/create_textflow()` 87

**mask** in `load_image()` 157

**masked** in `load_image()` 157

**masterpassword** in `begin_document()` 34

## **matchbox**

in `fit/info_textline()` and `add/create_textflow()` 87  
in various functions 112  
suboption for `createlastindent` in `fit_textflow()` 92

**maxchar** in `create_field()` and `create_fieldgroup()` 206

**maxcode** in `info_font()` 60

**maxlinewidth** keyword in `info_textflow()` 97

**maxlines** in `fit_textflow()` 93

**maxliney** keyword in `info_textflow()` 97

**maxspacing** in `add/create_textflow()` 86

**mediabox** in `begin/end_page_ext()` 42

**menuname** in `create_action()` 189

## **metadata**

in `begin/end_document()` 34  
in `begin/end_page_ext()` 42  
in `begin_template_ext()` 162  
in `load_font()` 55  
in `load_iccprofile()` 145  
in `load_image()` and `begin_template_ext()` 157  
in `open_pdi_page()` 172

**metricsfile** in `info_font()` 60

## **mimetype**

in `add_portfolio_file()` 212  
in `create_annotation()` 199  
suboption for attachments in `begin/end_document()` 33

**minfontsize** in `fit_textflow()` 93, 112

**mingapwidth** in `fit_textflow()` 93

**minlinecount** in `add/create_textflow()` 85

**minlinewidth** keyword in `info_textflow()` 97

**minliney** keyword in `info_textflow()` 97

**minrowheight** in `add_table_cell()` 101

**minspacing** in `add/create_textflow()` 86

## **mirroringx, mirroringy**

keywords in `info_image()` 161  
keywords in `info_pdi_page()` 174

## **miterlimit**

in `add_path_point()` 136  
in `create_gstate()` 125  
in several functions 120

**moddate** in `begin/end_document()` 34

**modeltree** suboption for `3Dactivate` in `create_annotation()` 201

**monospace**

in `info_font()` 60

in `load_font()` 55

**move** keyword in `add_path_point()` 136

**movieposter** in `create_annotation()` 199

**multiline** in `create_field()` and

`create_fieldgroup()` 207

**multiselect** in `create_field()` and

`create_fieldgroup()` 207

## N

### N

in `shading()` 151

suboption for shading option of several functions 121

**name**

in `add_path_point()` 137

in `add_portfolio_file()` 212

in `create_3dview()` 219

in `create_annotation()` 199

keyword in `info_matchbox()` 117

suboption for attachments in `begin/end_document()` 33

suboption for codepage in `info_font()` 59

suboption for feature in `info_font()` 60

suboption for matchbox 116

suboption for targetpath in `create_action()` 191

suboption for viewports in `begin/end_page_ext()` 222

**namelist** in `create_action()` 190

**newwindow** in `create_action()` 190

**nextline** in `add/create_textflow()` 87

**nextparagraph** in `add/create_textflow()` 87

**nofitlimit** in `add/create_textflow()` 86

**nonfullscreenpagemode** suboption for viewerpreferences in `begin/end_document()` 38

**normal** suboption for template in `create_annotation()` 200

**numcids** in `info_font()` 60

**numcopies** suboption for viewerpreferences in `begin/end_document()` 38

**numglyphs** in `info_font()` 61

**numpoints** in `info_path()` 139

**numusableglyphs** in `info_font()` 61

**numusedglyphs** in `info_font()` 61

## O

**objectstreams** in `begin_document()` 34

**offset**

suboption for shadow in `fit_textline()` 80

suboption for wrap in `fit_textflow()` 96

**offsetbottom, offsetleft, offsetright, offsettop**

suboptions for matchbox 116

**onpanel** in `define_layer()` 46

**opacity**

in `create_annotation()` 199

suboption for rendermode in `create_3dview()` 221

**opacityfill** in `create_gstate()` 125

**opacystroke** in `create_gstate()` 125

**open**

in `create_annotation()` 199

in `create_bookmark()` 210

**openmode** in `begin/end_document()` 34

**openrect** suboption for matchbox 116

**operation** in `create_action()` 190

**OPI-1.3** in `load_image()` and

`begin_template_ext()` 158

**OPI-2.0** in `load_image()` and

`begin_template_ext()` 158

**optimize** in `begin_document()` 35

**optimizeinvisible** in `load_font()` 55

**orientate**

in `create_annotation()` 199

in `create_field()` and `create_fieldgroup()` 207

in `fit_textflow()` 94

in various functions 112

**origin** keyword in `add_path_point()` 136

**outlineformat** in `info_font()` 61

**overline** in many functions 76

**overprintfill** in `create_gstate()` 125

**overprintmode** in `create_gstate()` 125

**overprintstroke** in `create_gstate()` 126

## P

### page

in `load_image()` 158

option in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 192

**pageelement** in `define_layer()` 46

**pageheight** keyword in `info_pdi_page()` 174

**pagelabel** suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 163

**pagelayout** in `begin/end_document()` 35

**pagenumber**

in `begin_page_ext()` 42

in `resume_page()` 44

suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 36

suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 163

suboption for targetpath in `create_action()` 191

**pages** suboption for separationinfo in `begin/end_page_ext()` 42

**pagewidth** keyword in `info_pdi_page()` 174

**parameters** in `create_action()` 190

**parent**

- in `begin_item()` 230
- in `create_bookmark()` 210
- in `set_layer_dependency()` 48

**parentname** in `create_annotation()` 199

**parindent** in `add/create_textflow()` 85

**passthrough** in `load_image()` 158

**password**

- in `add_portfolio_file()` 212
- in `create_field()` and `create_fieldgroup()` 207
- in `open_pdi_document` 168

**path**

- in `add_table_cell()` 101
- suboption for `textpath` in `fit_textline()` 81

**paths** suboption for `wrap` in `fit_textflow()` 96

**pdfa** in `begin_document()` 35

**pdfx** in `begin_document()` 35

**pdpage** in `add_table_cell()` 101

**pdusebox**

- in `open_pdi_page()` 172
- suboption for `reference` in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 163

**permissions** in `begin_document()` 35

**perpendiculardir** keyword in `info_textline()` 82

**picktraybypdfsize** suboption for `viewerpreferences` in `begin/end_document()` 38

**playmode** in `create_annotation()` 199

**polar** in `add_path_point()` 137

**polygons** suboption for `wrap` in `fit_textflow()` 96

**polylinelist** in `create_annotation()` 199

**popup** in `create_annotation()` 199

**portfolio** in `end_document()` 35

**position**

- in `create_field()` and `create_fieldgroup()` 207
- in various functions 112
- suboption for `template` in `create_annotation()` 200

**postscript** in `begin_template_ext()` 163

**predefcmap** in `info_font()` 61

**prefix**

- suboption for `fieldlist` in `add_portfolio_folder()` and `add_portfolio_file()` 213
- suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 37

**preserveradio** in `create_action()` 190

**printarea** suboption for `viewerpreferences` in `begin/end_document()` 38

**printclip** suboption for `viewerpreferences` in `begin/end_document()` 38

**printpagerange** suboption for `viewerpreferences` in `begin/end_document()` 38

**printscaling** suboption for `viewerpreferences` in `begin/end_document()` 38

**printsubtype** in `define_layer()` 46

**properties** in `begin_mc()` and `mc_point()` 232

**px, py** in `info_path()` 139

## R

**ro, r1** in `shading()` 151

**radians** in `add_path_point()` 137

**readfeatures** in `load_font()` 55

**readkerning** in `load_font()` 55

**readonly**

- in `create_annotation()` 199
- in `create_field()` and `create_fieldgroup()` 207

**readshaping** in `load_font()` 55

**recordsize** in `begin_document()` 35

**rectangle** keyword in `info_matchbox()` 117

**reference**

- in `begin_template_ext()` 162, 163
- in `open_pdi_page()` 172

**refpoint**

- in `fill_block()` 139
- in `fill_block()` and `info_path()` 112

**relation** suboption for `targetpath` in `create_action()` 191

**relative** in `add_path_point()` 137

**remove** for logging parameter 29

**removeunused** in `define_layer()` 46

**rendercolor** suboption for `rendermode` in `create_3dview()` 221

**renderingintent**

- in `create_gstate()` 126
- in `load_image()` 158

**rendermode** in `create_3dview()` 219

**repair** in `open_pdi_document` 168

**repeatcontent** in `add_table_cell()` 101

**replacedchars** in `info_textline()` 82

**replacementchar**

- in `info_font()` 61
- in `load_font()` 56

**replyto** in `create_annotation()` 199

**required** in `create_field()` and `create_fieldgroup()` 207

**requiredmode** in `open_pdi_document` 168

**resetfont** in `add/create_textflow()` 87

**resolution**

- suboption for the `barcode` option in `create_field()` and `create_fieldgroup()` 208

**resourcefile** in `set_option()` 21

**resx, resy** keywords in `info_image()` 161

**return**

- in `add/create_textflow()` 87
- in `add_table_cell()` 101

**returnatmark** in `fit_textflow()` 94

**returnreason**

- keyword in `info_table()` 106
- keyword in `info_textflow()` 97

**rewind**

- in `fit_table()` 104
- in `fit_textflow()` 94

**richtext** in `create_field()` and `create_fieldgroup()` 207

**right** option in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 192

**rightindent**  
in `add/create_textflow()` 85  
suboption for `createlastindent` in `fit_textflow()` 92

**rightline**, **rightliney** keywords in `info_textflow()` 97

**righttoleft** in `info_textline()` 82

**rolemap** in `begin_document()` 35

**rollover**  
suboption for `template` in `create_annotation()` 200

**rotate**  
in `begin/end_page_ext()` 42  
in `create_annotation()` 199  
in `fit_textflow()` 94  
in various functions 112  
keyword in `info_pdi_page()` 174  
suboption for `textpath` in `fit_textline()` 81

**round**  
in `add_path_point()` 137  
in `draw_path()` 138  
suboption for `matchbox` 116  
suboption for `textpath` in `fit_textline()` 81

**rowcount** keyword in `info_table()` 106

**rowheight** in `add_table_cell()` 101

**rowheightdefault** in `fit_table()` 105

**rowjoingroup** in `add_table_cell()` 102

**rowscalegroup** in `add_table_cell()` 101

**rowspan** in `add_table_cell()` 102

**RowSpan** in `begin_item()` 230

**rowsplit** keyword in `info_table()` 106

**ruler** in `add/create_textflow()` 85

## S

**scale**  
in various functions 113  
suboption for `textpath` in `fit_textline()` 81

**scalex**, **scaley** keywords in `info_textline()` 82

**schema** suboption for `portfolio` in `begin_document()` 214

**Scope** in `begin_item()` 230

**script**  
in `create_action()` 190  
in `load_3d()` 217  
in many functions 77  
suboption for `feature` in `info_font()` 60

**scriptlist** keyword in `info_textline()` 82

**scriptname** in `create_action()` 190

**scrollable** in `create_field()` and `create_fieldgroup()` 207

**search** in `begin/end_document()` 36

**searchpath** in `set_option()` 21

**separationinfo** in `begin_page_ext()` 42

**shading** in several functions 120

**shadow** in `fit/info_textline()` 80

**shaping** in many functions 77

**shapingsupport** in `info_font()` 61

**showborder**  
in `fit_textflow()` 94  
in various functions 113

**showcaption** in `create_annotation()` 200

**showcells** in `fit_table()` 105

**showcontrols** in `create_annotation()` 200

**showgrid** in `fit_table()` 105

**showtabs** in `fit_textflow()` 94

**shrinklimit**  
in `add/create_textflow()` 86  
in various functions 113

**shutdownstrategy** in `set_option()` 21

**singfont** in `info_font()` 61

**smoothness** in `create_gstate()` 126

**softmask** in `create_gstate()` 126

**sort** suboption for `portfolio` in `begin_document()` 214

**sorted** in `create_field()` and `create_fieldgroup()` 207

**soundvolume** in `create_annotation()` 200

**space** in `add/create_textflow()` 87

**spellcheck** in `create_field()` and `create_fieldgroup()` 207

**split**  
keyword in `info_textflow()` 97  
suboption for `portfolio` in `begin_document()` 215

**spotcolor** suboption for `separationinfo` in `begin/end_page_ext()` 42

**spotname** suboption for `separationinfo` in `begin/end_page_ext()` 42

**spreadlimit** in `add/create_textflow()` 87

**stamp**  
in `fit_textflow()` 94  
in various functions 113

**standardfont** in `info_font()` 61

**start**  
suboption for `labels` in `begin/end_document()` and `label` in `begin/end_page_ext()` 37  
suboption for `shading` option of several functions 121

**startcolor** in `shading()` 151

**startoffset** suboption for `textpath` in `fit_textline()` 81

**startx**, **starty** keywords in `info_textline()` 82

**stretch** in `begin_font()` 62

**strikeout** in many functions 76

**stringlimit** for logging parameter 29

**strips** keyword in `info_image()` 161

**stroke**  
in `draw_path()` 137, 138  
in `fit_table()` 105

**strokeadjust** in `create_gstate()` 126

**strokecolor**  
 in `add_path_point()` 136  
 in `create_field()` and `create_fieldgroup()` 207  
 in many functions 76  
 in several functions 120

**strokewidth** in many functions 76

**strongref** suboption for reference in  
`begin_template_ext()` and `open_pdi_page()`  
 164

**style** suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 37

**subject** in `create_annotation()` 200

**submitemptyfields** in `create_action()` 190

**submitname** in `create_field()` and  
`create_fieldgroup()` 207

**subpaths**  
 in `draw_path()` 138  
 suboption for `textpath` in `fit_textline()` 81

**subsetlimit** in `load_font()` 56

**subsetminsize** in `load_font()` 56

**subsetting** in `load_font()` 56

**supplement** in `info_font()` 61

**symbolfont** in `info_font()` 61

**symbology**  
 suboption for the barcode option in  
`create_field()` and `create_fieldgroup()` 208

## T

**tabalignchar** in `add/create_textflow()` 88

**tabalignment** in `add/create_textflow()` 85

**taborder**  
 in `begin/end_page_ext()` 42  
 in `create_field()` and `create_fieldgroup()` 207

**tagged** in `begin_document()` 36

**tagname** in `begin_item()` 230

**target**  
 in `create_action()` 190  
 suboption for reference in  
`begin_template_ext()`, `load_image()`, and  
`open_pdi_page()` 164

**targetbox** keyword in `info_image()` 161

**targetpath**  
 in `create_action()` 190  
 suboption for `targetpath` in `create_action()`  
 191

**targetx1, targety1, ..., targetx4, targety4**  
 keywords in `info_image()` 161

**tempdirname** in `begin_document()` 36

**tempfilenames** in `begin_document()` 36

**template**  
 in `create_annotation()` 200  
 in `load_image()` 158

**text** suboption for leader in `fit/info_textline()`  
 and `add/create_textflow()` 78

**textcolor** in `create_bookmark()` 210

**textendx, textendy** keywords in `info_textflow()`  
 97

**textflow**  
 in `add_table_cell()` 102  
 in `fill_textblock()` 183  
 suboption for `createrichtext` in  
`create_annotation()` 197

**textflowhandle** in `fill_textblock()` 183

**textformat** in many functions 75

**textheight** keyword in `info_textflow()` 97

**textknockout** in `create_gstate()` 126

**textlen** in `create_textflow()` 90

**textpath** in `fit/info_textline()` 80

**textrendering** in many functions 76

**textrise** in many functions 76

**textwidth** keyword in `info_textflow()` 98

**thumbnail**  
 in `add_portfolio_file()` 212  
 in `add_portfolio_folder()` 211

**Title** in `begin_item()` 230

**title** in `create_annotation()` 200

**toggle** in `create_fieldgroup()` 207

**tolerance** suboption for `textpath` in `fit_textline()`  
 81

**toolbar**  
 suboption for `3Dactivate` in  
`create_annotation()` 201

**tooltip** in `create_field()` and `create_fieldgroup()`  
 207

**top** option in `add_nameddest()` and suboption for  
 destination in `create_action()`,  
`create_annotation()`, `create_bookmark()` and  
`begin/end_document()` 192

**topdown**  
 in `begin_page_ext()` 42  
 in `begin_template_ext()` 162

**topindex** in `create_field()` and `create_fieldgroup()`  
 207

**transition**  
 in `begin/end_page_ext()` 43  
 in `create_action()` 191

**transparencypgroup**  
 in `begin/end_page_ext()` 43  
 in `begin_template_ext()` 162  
 in `open_pdi_page()` 171

**trimbox** in `begin/end_page_ext()` 43

**truncatetrailingwhitespace**  
 in `fit_textflow()` 94

**type**  
 in `create_3dview()` 219  
 in `load_3d()` 218  
 option in `add_nameddest()` and suboption for  
 destination in `create_action()`,

*create\_annotation()*, *create\_bookmark()* and *begin/end\_document()* 193  
suboption for *custom* in *create\_annotation()* 197  
suboption for *fieldlist* in *add\_portfolio\_folder()* and *add\_portfolio\_file()* 213  
suboption for *properties* in *begin\_mc()* and *mc\_point()* 232  
suboption for *rendermode* in *create\_3dview()* 221  
suboption for *shading* option of several functions 121  
suboption for the *coords* and *displaycoords* suboptions of *georeference* 223

## U

*U3Dpath* in *create\_3dview()* 219  
*underline* in many functions 76  
*underlineposition* many functions 77  
*underlinewidth* in many functions 77  
*unicode* in *info\_font()* 58, 61  
*unicodefont* in *info\_font()* 61  
*unicodemap* in *load\_font()* 56  
*unisonselect* in *create\_fieldgroup()* 207  
*unknownchars* in *info\_textline()* 82  
*unmappedchars*  
in *info\_font()* 61  
in *info\_textline()* 82  
*uri* in *begin/end\_document()* 36  
*url* in *create\_action()* 191  
*urls* in *load\_iccprofile()* 145  
*usage* in *load\_iccprofile()* 145  
*used* keyword in *info\_textflow()* 98  
*usedglyph* in *info\_font()* 61  
*usematchbox* in *create\_annotation()* 200  
*usematchboxes* suboption for *wrap* in *fit\_textflow()* 96  
*usercoordinates*  
in *create\_annotation()* 200  
in *create\_field()* and *create\_fieldgroup()* 208  
*userpassword* in *begin\_document()* 36  
*userunit*  
in *begin/end\_page\_ext()* 43  
suboption for *createrichtext* in *create\_annotation()* 197

## V

*value*  
suboption for *custom* in *create\_annotation()* 197  
suboption for *fieldlist* in *add\_portfolio\_folder()* and *add\_portfolio\_file()* 213  
suboption for *properties* in *begin\_mc()* and *mc\_point()* 232  
*variantname* in *set\_layer\_dependency()* 48

*vertboxgap* keyword in *info\_table()* 106  
*vertical*  
in *info\_font()* 61  
in *load\_font()* 56  
*verticalalign* in *fit\_textflow()* 95  
*vertshrinking* keyword in *info\_table()* 106  
*vertshrinklimit* in *fit\_table()* 105  
*viewarea* suboption for *viewerpreferences* in *begin/end\_document()* 38  
*viewclip* suboption for *viewerpreferences* in *begin/end\_document()* 38  
*viewerpreferences* in *begin\_document()* and *end\_document()* 36  
*viewports* in *begin/end\_page\_ext()* 43  
*views* in *load\_3d()* 218  
*visiblelayers* in *set\_layer\_dependency()* 48

## W

*weight*  
in *begin\_font()* 62  
in *info\_font()* 61  
*wellformed* keyword in *info\_textline()* 82  
*width*  
in *begin/end\_page\_ext()* 43  
in *load\_image()* 158  
keyword in *info\_image()* 161  
keyword in *info\_matchbox()* 117  
keyword in *info\_path()* 139  
keyword in *info\_pdi\_page()* 175  
keyword in *info\_table()* 106  
keyword in *info\_textline()* 82  
*widthsonly* in *begin\_font()* 63  
*willembd* in *info\_font()* 61  
*willsubset* in *info\_font()* 61  
*windowposition* in *create\_annotation()* 200  
*windowyscale* in *create\_annotation()* 201  
*wkt* suboption for the *coords* and *displaycoords* suboptions of *georeference* 223  
*wordspacing* in many functions 77  
*worldpoints* suboption for *georeference* 223  
*wrap* in *fit\_textflow()* 95  
*writingdirx*, *writingdiry* keywords in *info\_textline()* 82

## X

*x1, y1, ..., x4, y4*  
keywords in *info\_image()* 161  
keywords in *info\_matchbox()* 117  
keywords in *info\_path()* 139  
keywords in *info\_pdi\_page()* 175  
keywords in *info\_table()* 106  
keywords in *info\_textflow()* 98  
*xadvancedlist* in *fit/info\_textline()* 80  
*xheight*  
in *info\_font()* 61  
in *load\_font()* 56  
keyword in *info\_textline()* 82

***xsymheight***

*suboption for the barcode option in  
create\_field() and create\_fieldgroup() 208*

***xsymwidth***

*suboption for the barcode option in  
create\_field() and create\_fieldgroup() 208*

***xvertline*** keyword in *info\_table()* 106

**Y**

***yhorline*** keyword in *info\_table()* 106

***yposition*** suboption for leader in *fit/  
info\_textline()* and *add/create\_textflow()* 78

**Z*****zoom***

*in add\_nameddest() and suboption for  
destination in create\_action(),  
create\_annotation(), create\_bookmark() and  
begin/end\_document() 193  
in create\_annotation() 201  
in define\_layer() 46*



# D Revision History

<b>Date</b>	<b>Changes</b>
June 11, 2013	▶ Various updates and corrections for PDFlib 8.0.6
October 23, 2012	▶ Various updates and corrections for PDFlib 8.0.5
December 22, 2011	▶ Various updates and corrections for PDFlib 8.0.4
July 11, 2011	▶ Various updates and corrections for PDFlib 8.0.3
December 09, 2010	▶ Various updates and corrections for PDFlib 8.0.2
September 22, 2010	▶ Various updates and corrections for PDFlib 8.0.1p7
April 13, 2010	▶ Various updates and corrections for PDFlib 8.0.1
December 04, 2009	▶ Updates for PDFlib 8.0.0
April 20, 2010	▶ Minor corrections for PDFlib 7.0.5
March 13, 2009	▶ Various updates and corrections for PDFlib 7.0.4
February 13, 2008	▶ Various updates and corrections for PDFlib 7.0.3
August 08, 2007	▶ Various updates and corrections for PDFlib 7.0.2
March 09, 2007	▶ Various updates and corrections for PDFlib 7.0.1
October 03, 2006	▶ Updates and restructuring for PDFlib 7.0.0; split the manual in tutorial and API reference
February 15, 2007	▶ Various updates and corrections for PDFlib 6.0.4
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

<b>Date</b>	<b>Changes</b>
<i>Feb. 20, 2000</i>	▶ <i>Changes for PDFlib 3.0</i>
<i>Aug. 2, 1999</i>	▶ <i>Minor changes and additions for PDFlib 2.01</i>
<i>June 29, 1999</i>	▶ <i>Separate sections for the individual language bindings</i> ▶ <i>Extensions for PDFlib 2.0</i>
<i>Feb. 1, 1999</i>	▶ <i>Minor changes for PDFlib 1.0 (not publicly released)</i>
<i>Aug. 10, 1998</i>	▶ <i>Extensions for PDFlib 0.7 (only for a single customer)</i>
<i>July 8, 1998</i>	▶ <i>First attempt at describing PDFlib scripting support in PDFlib 0.6</i>
<i>Feb. 25, 1998</i>	▶ <i>Slightly expanded the manual to cover PDFlib 0.5</i>
<i>Sept. 22, 1997</i>	▶ <i>First public release of PDFlib 0.4 and this manual</i>

# Index

Note that parameters and options are listed in separate appendices.

## A

*action lists in option lists* 13  
*alignment (position option)* 112  
*All spot color name* 143  
*any scope* 18  
*Author field* 225

## B

*Bézier curve* 131  
*Boolean values in option lists* 11

## C

*circles in option lists* 14  
*CMYK color* 141  
*cmyk keyword* 13  
*color functions* 141  
*color in option lists* 12  
*Creator field* 225  
*curves in option lists* 15

## D

*dash pattern for lines* 122  
*document and page functions* 31  
*document information fields* 225  
*document scope* 18  
*Dublin Core* 225

## F

*fast Web view* 34  
*float and integer values in option lists* 11  
*floats in option lists* 11  
*font scope* 18  
*fontsize in option lists* 12  
*function scopes* 17

## G

*glyph scope* 18  
*graphics functions* 119  
*gray keyword* 13

## H

*handles in option lists* 11

## I

*ICC Profiles* 145

*ICC-based color* 141  
*iccbasedcmyk keyword* 13  
*iccbasedgray keyword* 13  
*iccbasedrgb keyword* 13  
*image functions* 153  
*import functions for PDF (PDI)* 167  
*indexed color* 142  
*info fields* 225  
*inline option lists for Textflows* 90  
*inner cell box for table cells* 101  
*invisible text* 76

## K

*Keywords field* 225  
*keywords in option lists* 11

## L

*lab keyword* 13  
*landscape mode* 41  
*licence* 22  
*license* 22  
*linearized PDF* 34  
*lines in option lists* 14  
*lines: dashed and patterned* 122  
*list values in option lists* 8  
*logging* 29

## M

*metadata* 227  
*mirroring* 127

## N

*nested option lists* 8  
*None spot color name* 143  
*null scope* 18  
*numbers in option lists* 11

## O

*object scope* 18  
*option list syntax* 7  
*outline text* 76

## P

*page scope* 18  
*page size formats* 40  
*parameter handling functions* 19  
*path painting and clipping* 134

*path scope* 18  
*pattern keyword* 13  
*pattern scope* 18  
*pCOS functions* 167, 177  
*PDF import functions (PDI)* 167  
*PDF/A or PDF/X output intent* 176  
*PDF\_activate\_item()* 230  
*PDF\_add\_nameddest()* 192  
*PDF\_add\_path\_point()* 136  
*PDF\_add\_portfolio\_folder()* 211  
*PDF\_add\_table\_cell()* 99  
*PDF\_add\_textflow()* 83  
*PDF\_add\_thumbnail()* 165  
*PDF\_align()* 128  
*PDF\_arc()* 131, 132  
*PDF\_arcn()* 132  
*PDF\_begin\_document()* 31  
*PDF\_begin\_font()* 62  
*PDF\_begin\_glyph()* 63  
*PDF\_begin\_item()* 228  
*PDF\_begin\_layer()* 48  
*PDF\_begin\_mc()* 232  
*PDF\_begin\_page\_ext()* 40, 41  
*PDF\_begin\_pattern* 149  
*PDF\_begin\_template\_ext()* 162  
*PDF\_circle()* 131, 132  
*PDF\_clip()* 135  
*PDF\_close\_font()* 57  
*PDF\_close\_image()* 159  
*PDF\_close\_pdi\_document()* 169  
*PDF\_close\_pdi\_page()* 173  
*PDF\_closepath()* 133  
*PDF\_closepath\_fill\_stroke()* 135  
*PDF\_closepath\_stroke()* 134  
*PDF\_concat()* 128  
*PDF\_continue\_text()* 69  
*PDF\_continue\_textz()* 69  
*PDF\_create\_3dview()* 218  
*PDF\_create\_action()* 187  
*PDF\_create\_annotation()* 194  
*PDF\_create\_bookmark()* 209  
*PDF\_create\_field()* 202  
*PDF\_create\_fieldgroup()* 203  
*PDF\_create\_gstate()* 125  
*PDF\_create\_pvf()* 25  
*PDF\_create\_textflow()* 89  
*PDF\_curveto()* 131  
*PDF\_define\_layer()* 45  
*PDF\_delete()* 24  
*PDF\_delete\_dl()* 24  
*PDF\_delete\_path()* 139  
*PDF\_delete\_pvf()* 26  
*PDF\_delete\_table()* 107  
*PDF\_delete\_textflow()* 98  
*PDF\_draw\_path()* 137  
*PDF\_encoding\_set\_char()* 65  
*PDF\_end\_document()* 32  
*PDF\_end\_font()* 63  
*PDF\_end\_glyph()* 64  
*PDF\_end\_item()* 230  
*PDF\_end\_layer()* 49  
*PDF\_end\_mc()* 232  
*PDF\_end\_pattern()* 149  
*PDF\_end\_template\_ext()* 164  
*PDF\_endpath()* 135  
*PDF\_fill()* 134  
*PDF\_fill\_imageblock()* 184  
*PDF\_fill\_pdfblock()* 185  
*PDF\_fill\_stroke()* 134  
*PDF\_fill\_textblock()* 182  
*PDF\_fit\_image()* 159  
*PDF\_fit\_pdi\_page()* 173  
*PDF\_fit\_table()* 102  
*PDF\_fit\_textflow()* 91  
*PDF\_fit\_textline()* 79  
*PDF\_get\_apiname()* 28  
*PDF\_get\_buffer()* 39  
*PDF\_get\_errmsg()* 27  
*PDF\_get\_errnum()* 27  
*PDF\_get\_opaque()* 28  
*PDF\_get\_parameter()* 19  
*PDF\_get\_value()* 19  
*PDF\_info\_font()* 58  
*PDF\_info\_image()* 160  
*PDF\_info\_matchbox()* 116  
*PDF\_info\_path()* 138  
*PDF\_info\_pdi\_page()* 173  
*PDF\_info\_table()* 106  
*PDF\_info\_textflow()* 95  
*PDF\_info\_textline()* 81  
*PDF\_initgraphics()* 124  
*PDF\_lineto()* 130  
*PDF\_load\_3ddata()* 217  
*PDF\_load\_font()* 51  
*PDF\_load\_iccprofile()* 145  
*PDF\_load\_image()* 154  
*PDF\_makespotcolor()* 143  
*PDF\_mc\_point()* 233  
*PDF\_moveto()* 130  
*PDF\_new()* 23  
*PDF\_new\_dl()* 23  
*PDF\_newz()* 23  
*PDF\_open\_pdi\_callback()* 169  
*PDF\_open\_pdi\_document()* 167  
*PDF\_open\_pdi\_page()* 171  
*PDF\_pcos\_get\_number()* 177  
*PDF\_pcos\_get\_stream()* 178  
*PDF\_pcos\_get\_string()* 177  
*PDF\_process\_pdi()* 176  
*PDF\_rect()* 133  
*PDF\_restore()* 125  
*PDF\_resume\_page()* 44  
*PDF\_rotate()* 127  
*PDF\_save()* 124  
*PDF\_scale()* 127  
*PDF\_set\_gstate()* 126

*PDF\_set\_info()* 225  
*PDF\_set\_info2()* 225  
*PDF\_set\_layer\_dependency()* 46  
*PDF\_set\_option()* 20  
*PDF\_set\_parameter()* 20  
*PDF\_set\_text\_pos()* 67  
*PDF\_set\_value()* 19  
*PDF\_setcolor()* 142  
*PDF\_setdash()* 122  
*PDF\_setdashpattern()* 122  
*PDF\_setflat()* 122  
*PDF\_setfont()* 67  
*PDF\_setlinecap()* 123  
*PDF\_setlinejoin()* 123  
*PDF\_setlinewidth()* 123  
*PDF\_setmatrix()* 129  
*PDF\_setmiterlimit()* 123  
*PDF\_shading()* 150  
*PDF\_shading\_pattern()* 150  
*PDF\_shfill()* 150  
*PDF\_show()* 68  
*PDF\_show\_xy()* 69  
*PDF\_show\_xy2()* 69  
*PDF\_show2()* 68  
*PDF\_skew()* 128  
*PDF\_stringwidth()* 70  
*PDF\_stringwidth2()* 70  
*PDF\_stroke()* 134  
*PDF\_suspend\_page()* 44  
*PDF\_translate()* 127  
*PDF\_utf16\_to\_utf32()* 73  
*PDF\_utf16\_to\_utf8()* 71  
*PDF\_utf32\_to\_utf16()* 72  
*PDF\_utf32\_to\_utf8()* 73  
*PDF\_utf8\_to\_utf16()* 71, 72, 73  
*PDF\_utf8\_to\_utf32()* 72  
*PDF\_xshow()* 68  
*PDFlib Personalization Server (PPS)* 181  
*PDI (PDF import)* 167  
*polylines in option lists* 14  
*PPS (PDFlib Personalization Server)* 181

## R

*raster image functions* 153

*rectangles in option lists* 14  
*reflection* 127  
*RGB color* 141  
*rgb keyword* 13

## S

*scopes* 17  
*separation color space* 141  
*setup functions* 22  
*skewing* 128  
*spot color (separation color space)* 141  
*spot keyword* 13  
*spotname keyword* 13  
*standard page sizes* 40  
*string index* 22  
*strings in option lists* 10  
*Subject field* 225  
*subscript* 76  
*superscript* 76  
*syntax of option lists* 7

## T

*table formatting* 99  
*template scope* 18  
*text functions* 51  
*Textflow: inline option lists* 90  
*thumbnails* 165  
*Title field* 225  
*Trapped field* 225

## U

*Unichar values in option lists* 10  
*Unicode ranges in option lists* 11  
*userlog* 29

## W

*web-optimized PDF* 34

## X

*XMP metadata* 227

**PDFlib GmbH**

Franziska-Bilek-Weg 9  
80339 München, Germany  
www.pdflib.com  
phone +49 • 89 • 452 33 84-0  
fax +49 • 89 • 452 33 84-99

If you have questions check the PDFlib mailing list  
and archive at [tech.groups.yahoo.com/group/pdflib](http://tech.groups.yahoo.com/group/pdflib)

**Licensing contact**

[sales@pdflib.com](mailto:sales@pdflib.com)

**Support**

[support@pdflib.com](mailto:support@pdflib.com) (*please include your license number*)

