

# PDFlib, PDFlib+PDI, PPS

A library for generating PDF on the fly  
PDFlib 9.0.4

## API Reference

For use with C, C++, Cobol, COM, Java, .NET, Objective-C,  
Perl, PHP, Python, REALbasic/Xojo, RPG, Ruby



Copyright © 1997–2015 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  
[groups.yahoo.com/neo/groups/pdflib/info](mailto:groups.yahoo.com/neo/groups/pdflib/info)

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 © 1998-2012 Glenn Randers-Pehrson

Zlib compression library, Copyright © 1995-2012 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 ([ey@cryptsoft.com](mailto:ey@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 © 1995-2012 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 Programming Concepts 7

- 1.1 Option Lists 7
  - 1.1.1 Syntax 7
  - 1.1.2 Simple Data Types 10
  - 1.1.3 Fontsize and Action Data Types 12
  - 1.1.4 Color Data Type 13
  - 1.1.5 Geometric Data Types 15
- 1.2 Function Scopes 17
- 1.3 Logging 18

## 2 General Functions 21

- 2.1 Exception Handling 21
- 2.2 Unicode Conversion 23
- 2.3 Global Options 25
- 2.4 Creating and Deleting PDFlib Objects 32
- 2.5 PDFlib Virtual File System (PVF) 34
- 2.6 PDF Object Creation API (POCA) 37

## 3 Document and Page Functions 41

- 3.1 Document Functions 41
- 3.2 Fetching PDF Documents from Memory 51
- 3.3 Page Functions 52
- 3.4 Layers 57

## 4 Font and Text Functions 63

- 4.1 Font Handling 63
- 4.2 Text Filter and Appearance Options 75
- 4.3 Simple Text Output 80
- 4.4 User-defined (Type 3) Fonts 84
- 4.5 User-defined 8-Bit Encodings 87

## 5 Text and Table Formatting 89

- 5.1 Single-Line Text with Textlines 89
- 5.2 Multi-Line Text with Textflows 95
- 5.3 Table Formatting 112

## 6 Object Fitting and Matchboxes 123

- 6.1 Object Fitting 123

6.2 Matchboxes 131

## **7 Graphics Functions 135**

- 7.1 Graphics Appearance Options 135
- 7.2 Graphics State 138
- 7.3 Coordinate System Transformations 142
- 7.4 Path Construction 145
- 7.5 Painting and Clipping 149
- 7.6 Path Objects 151

## **8 Color Functions 157**

- 8.1 Setting Color 157
- 8.2 ICC Profiles 160
- 8.3 Patterns and Shadings 162

## **9 Image, SVG and Template Functions 167**

- 9.1 Images 167
- 9.2 SVG Graphics 175
- 9.3 Templates 181
- 9.4 CommonXObject Options 183

## **10 PDF Import (PDI) and pCOS Functions 187**

- 10.1 Document Functions 187
- 10.2 Page Functions 191
- 10.3 Other PDI Processing 197
- 10.4 pCOS Functions 199

## **11 Block Filling Functions (PPS) 203**

- 11.1 Rectangle Options for Block Filling Functions 203
- 11.2 Textline and Textflow Blocks 204
- 11.3 Image Blocks 206
- 11.4 PDF Blocks 207
- 11.5 Graphics Blocks 208

## **12 Interactive Features 209**

- 12.1 Bookmarks 209
- 12.2 Annotations 211
- 12.3 Form Fields 219
- 12.4 Actions 226
- 12.5 Named Destinations 231

12.6 PDF Packages and Portfolios 233

12.7 Geospatial Features 238

## **13 Multimedia Features 241**

13.1 3D Artwork 241

13.2 Asset and Rich Media Features (Flash) 247

## **14 Document Interchange 255**

14.1 Document Information Fields 255

14.2 XMP Metadata 257

14.3 Tagged PDF 258

14.4 Marked Content 264

14.5 Document Part Hierarchy 266

**A List of all API Functions 269**

**B List of all Options and Keywords 271**

**C Revision History 287**

**Index 289**



# 1 Programming Concepts

## 1.1 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.

.NET language binding: C# programmers should keep in mind that the *AppendFormat()* *StringBuilder* method uses the { and } braces to represent format items which will be replaced by the string representation of arguments. On the other hand, the *Append()* method does not impose any special meaning on the brace characters. Since the option list syntax makes use of the brace characters, care must be taken in selecting the *AppendFormat()* or *Append()* method appropriately.

### 1.1.1 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.xml}
```

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):

```
position={10 bottom}
```

List containing one rectangle:

```
boxes={{10 20 30 40}}
```

List containing two polylines with percentages:

```
polygons = {{10 20 40 60 90 120}} {12 87 34 98 34% 67% 34% 7%}}
```

**Common traps and pitfalls.** This paragraph lists some common errors regarding option list syntax.

Braces are not separators; the following is wrong:

```
key1 {value1}key2 {value2}                WRONG!
```

This will trigger the error message *Unknown option 'value2'*. Similarly, the following are wrong since the separators are missing:

```
key{value}                                WRONG!  
key={{value1}{value2}}                   WRONG!
```

Braces must be balanced; the following is wrong (see below for unquoted string syntax):

```
key={open brace }                         WRONG!
```

This will trigger the error message *Braces aren't balanced in option list 'key={open brace }'*. A single brace as part of a string must be preceded by an additional backslash character:

```
key={closing brace \} and open brace \{}   CORRECT!
```

A backslash at the end of a string value must be preceded by another backslash if it is followed by a closing brace character:

```
key={value\  
key={value\\}                             CORRECT!
```

**Unquoted string values in option lists.** In the following situations conflicts between the characters in an option value and optlist syntax characters may arise:

- ▶ Passwords may contain unbalanced braces, backslashes and other special characters
- ▶ Japanese SJIS filenames in option lists (reasonable only in non-Unicode-capable language bindings)
- ▶ Supplying JavaScript code in options is problematic due to the use of { and } braces

In order to provide a simple mechanism for supplying arbitrary text or binary data which does not interfere with option list syntax elements, unquoted option values can be supplied along with a length specifier in the following syntax variants:

```
key[n]=value  
key[n]={value}
```

The decimal number *n* represents the following:

- ▶ in Unicode-capable language bindings: the number of UTF-16 code units
- ▶ in non-Unicode aware language bindings: the number of bytes comprising the string

The braces around the string value are optional, but strongly recommended. They are required for strings starting with a space or other separator character. Braces, separators and backslashes within the string value are taken literally without any special interpretation.

Example for specifying a 7-character password containing space and brace characters. The whole string is surrounded by braces which are not part of the option value:

```
password[7]={ ab}c d}
```

If an option value in a nested option list is provided with a length count, the enclosing option list must also supply a length count, e.g.

```
fitannotation[34]={contents[19]={this is a brace '}'}}
```

## 1.1.2 Simple Data 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 option *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 is interpreted as a [EBCDIC-]UTF-8 string.

**Unichar.** A Unichar is a single Unicode value where several syntax variants are supported: decimal values  $\geq 10$  (e.g. 173), hexadecimal values prefixed with *x*, *X*, *ox*, *oX*, or *U+* (*xAD*, *oxAD*, *U+ooAD*), numerical references, character references, and glyph name references but without the '&' and ';' decoration (*shy*, *#xAD*, *#173*). Alternatively, literal characters can be supplied. Examples:

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

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

<code>replacementchar=3</code>	(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*:

<code>embedding</code>	(equivalent to <code>embedding=true</code> )
<code>noembedding</code>	(equivalent to <code>embedding=false</code> )

**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, ICC profiles or actions. Technically these are integer values which have been returned earlier by an API function. For example, an image handle is returned by *PDF\_load\_image()*. 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 not be supplied with a graphics handle, although both handles are integer types.

### 1.1.3 Fontsize and Action Data Types

**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}
```

**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

Table 1.1 Suboptions for options of type fontsize

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

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 }
```

## 1.1.4 Color Data Type

**Overview of color spaces.** You can specify the colors for filling and stroking paths and text characters. Colors can 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.

- ▶ *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.
- ▶ *iccbased* (not for `PDF_setcolor()`) and *iccbasedgray/rgb/cmyk*: ICC-based colors are based on an ICC profile.
- ▶ *spotname*: name of a predefined spot color and a tint value (percentage) in the range 0=no color to 1=maximum intensity.

Alternatively, the name of a custom spot color, a tint value (percentage), and an alternate representation in one of the other color spaces above.

- ▶ *spot*: handle for a predefined or custom spot color and a tint value (percentage).
- ▶ *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. Patterns can be created with `PDF_begin_pattern_ext()` or `PDF_shading_pattern()` and are identified by a pattern handle.

The default color for stroke and fill operations is black. The color space for this default color is selected automatically to match PDF/X and PDF/A color requirements.

*Note* *Shadings (smooth blends) provide a gradual transition between two colors. They can be created with `PDF_shading()`.*

**Color options.** Color options 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.

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

In the first form all valid color names from SVG 1.1 can be supplied directly to specify an RGB color or an sRGB color if the sRGB ICC profile has been selected, e.g.

```
strokecolor=pink
```

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

[www.w3.org/TR/SVG11/types.html#ColorKeywords](http://www.w3.org/TR/SVG11/types.html#ColorKeywords)

In the second form a hash '#' character followed by any 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 color space keyword and a list with a variable number of float values depending on the particular color space. Color space keywords are the same as for `PDF_setcolor()` (see Section 8.1, »Setting Color«, page 157). Table 1.2 contains specific descriptions and examples. As detailed in the respective function descriptions, a particular option list may support only a subset of the color space keywords.

*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
<i>gray</i>	single float value for the grayscale color space	{ gray 0.5 }
<i>rgb</i>	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
<i>cmyk</i>	four float values for the CMYK color space	{ cmyk 0 1 0 0 }
<i>lab</i>	three float values for the Lab color space	{ lab 100 50 30 }
<i>spot</i>	spot color handle and a float specifying the tint value	{ spot 1 0.8 }

Table 1.2 Keywords for the color data type in option lists

keyword	additional values	example
<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>iccbased</b>	ICC profile handle or keyword srgb, plus 1, 3 or 4 color values depending on the type of ICC profile (gray, RGB, or CMYK). The srgb keyword must not be used in document scope.	{ iccbased <handle> 0.5 } { iccbased <handle> 0 0 0.75 } { iccbased srgb 0 0 0.75 } { iccbased <handle> 0 0 0.3 1 }
<b>iccbasedgray</b>	single float value referring to an ICC profile selected with the option iccprofilegray	{ iccbasedgray 0.5 }
<b>iccbasedrgb</b>	three float values value referring to an ICC profile selected with the option iccprofilergb	{ iccbasedrgb 1 0 0 }
<b>iccbasedcmyk</b>	four float values value referring to an ICC profile selected with the option iccprofilecmyk	{ iccbasedcmyk 0 1 0 0 }
<b>pattern</b>	pattern handle	{ pattern 1 }
<b>none</b>	specifies the absence of color	none

### 1.1.5 Geometric Data 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:

```
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.



## 1.2 Function Scopes

PDFlib applications must obey certain structural rules which are easy to understand. For example, you obviously begin a document before ending it. PDFlib enforces correct ordering of function calls with a strict scoping system. The scope definitions can be found in Table 1.3. All API function descriptions specify the allowed scope for each function. Calling a function outside of the allowed scopes results in an exception. You can query the current scope with the *scope* keyword of `PDF_get_option()`.

Table 1.3 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> , <code>PDF_rect()</code> , <code>PDF_ellipse()</code> or <code>PDF_elliptical_arc()</code> ; terminated by any of the functions in Section 7.5, »Painting and Clipping«, page 149
<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_ext()</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_ext()</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>	during the lifetime of the PDFlib object, but outside of document scope; in the C and Cobol language bindings between <code>PDF_new()</code> and <code>PDF_delete()</code> , but outside of document scope

## 1.3 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. Logging options can be supplied in the following ways:

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

```
p.set_option("logging={filename=trace.log remove}")
```
- ▶ In an environment variable called `PDFLIBLOGGING`. This will activate the logging output starting with the very first call to one of the API functions.

Table 1.4 Suboptions for the logging option

<b>option</b>	<b>description</b>
<b>(empty list)</b>	Enable log output
<b>disable</b>	(Boolean) Disable logging output
<b>enable</b>	(Boolean) Enable logging output
<b>filename</b>	(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: pdflog                   on z/OS PDFlib.log               on Mac and iSeries \\PDFlib.log             on Windows /tmp/PDFlib.log         on all other systems The log file name can alternatively be supplied in an environment variable called <code>PDFLIBLOGFILE</code> .
<b>flush</b>	(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>
<b>includepid</b>	(Boolean; not on MVS) Include the process id in the log file name. This should be enabled if multiple processes use the same log file name. Default: <code>false</code>
<b>includetid</b>	(Boolean; not on MVS) Include the thread id in the log file name. This should be enabled if multiple threads in the same process use the same log file name. Default: <code>false</code>
<b>includeoid</b>	(Boolean; not on MVS) Include the object id in the log file name. This should be enabled if multiple PDFlib objects in the same thread use the same log file name. Default: <code>false</code>
<b>remove</b>	(Boolean) If <code>true</code> , an existing log file will be deleted before writing new output. Default: <code>false</code>
<b>removeon-success</b>	(Boolean) Remove the generated log file in <code>PDF_delete()</code> unless an exception occurred. This may be useful for analyzing occasional problems in multi-threaded applications or problems which occur only sporadically. It is recommended to combine this option with <code>includepid</code> / <code>includetid</code> / <code>includeoid</code> as appropriate.
<b>stringlimit</b>	(Integer) Limit for the number of characters per line, or 0 for unlimited. Default: 0

Table 1.4 Suboptions for the logging option

option	description
<b>classes</b>	<p>(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}):</p>
<b>api</b>	<p>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).</p>
<b>filesearch</b>	<p>Log all attempts related to locating files via SearchPath or PVF.</p>
<b>resource</b>	<p>Log all attempts at locating resources via Windows registry, UPR definitions as well as the results of the resource search.</p>
<b>tagging</b>	<p>Structure element (tag) operations</p>
<b>user</b>	<p>User-specified logging output supplied with the userlog option.</p>
<b>warning</b>	<p>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 <code>PDF_get_errmsg()</code>, and the reason for all failed attempts at opening a file (searching for a file in searchpath) will also be logged.</p>



# 2 General Functions

## 2.1 Exception Handling

Table 2.1 details the relevant option for this section. This option is supported by many functions as indicated in the corresponding option list descriptions. It can also be supplied as global option to `PDF_set_option()` (see Section 2.3, »Global Options«, page 25).

Table 2.1 Exception-related option for `PDF_set_option()`

key	explanation
<b>errorpolicy</b>	(Keyword) Controls the behavior of various functions in case of an error. The global option <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> ): <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 C# **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, .NET, PHP and REALbasic this function is also available as `get_errnum()` in the `PDFlibException` object.

---

C++ Java C# **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: o) error code, but before calling any other function except those listed in this section.

*Bindings* In C++, Java, Objective-C, .NET, PHP and REALbasic this function is also available as *get\_errmsg()* in the *PDFlibException* object.

---

C++ Java C# **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: o) error code, but before calling any other function except those listed in this section.

*Bindings* In C++, Java, Objective-C, .NET, PHP and REALbasic 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.

## 2.2 Unicode Conversion

---

C++ `string convert_to_unicode(string inputformat, string input, string optlist)`

Java `string convert_to_unicode(string inputformat, byte[] input, string optlist)`

Perl PHP `string convert_to_unicode(string inputformat, string input, string optlist)`

C `const char *PDF_convert_to_unicode(PDF *p, const char *inputformat, const char *input, int inputlen, int *outputlen, const char *optlist)`

---

Convert a string in an arbitrary encoding to a Unicode string in various formats.

**inputformat** Unicode text format or encoding name specifying interpretation of the input string:

- ▶ Unicode text formats: *utf8, ebcdicutf8, utf16, utf16le, utf16be, utf32*
- ▶ Only if the *font* option is specified: *builtin, glyphid*
- ▶ All internally known 8-bit encodings, encodings available on the host system, and the CJK encodings *cp932, cp936, cp949, cp950*
- ▶ The keyword *auto* specifies the following behavior: if the input string contains a UTF-8 or UTF-16 BOM it will be used to determine the appropriate format, otherwise the current system codepage is assumed.

**input** String (in COM: Variant; in REALbasic: MemoryBlock) to be converted to Unicode.

**inputlen** (C language binding only) Length of the input string in bytes. If *inputlen=0* a null-terminated string must be provided.

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

**optlist** An option list specifying options for input interpretation and Unicode conversion:

- ▶ Text filter options according to Table 4.6: *charref, escapesquence*
- ▶ Unicode conversion options according to Table 2.2: *bom, errorpolicy, font, inflate, outputformat*

**Returns** A Unicode string created from the input string according to the specified parameters and options. If the input string does not conform to the specified input format (e.g. invalid UTF-8 string) an empty output string will be returned if *errorpolicy=return*, and an exception will be thrown if *errorpolicy=exception*.

**Details** This function may be useful for general Unicode string conversion. It is provided for the benefit of users who work in environments without suitable Unicode converters.

**Scope** any

**Bindings** C binding: the returned strings will be stored in a ring buffer with up to 10 entries. If more than 10 strings are converted, the 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.

Non-Unicode-capable language bindings: this function can be used to create name strings and option lists in non-Unicode aware language bindings. It creates the required BOM with the options *bom=optimize* and *outputformat=utf8*.

C++ binding: The parameters *inputformat* and *optlist* must be passed as *wstrings* as usual, while *input* and returned data must have type *string*.

Table 2.2 Options for *PDF\_convert\_to\_unicode()*

<b>option</b>	<b>description</b>
<b>bom</b>	(Keyword; will be ignored for <i>outputformat=utf32</i> ) Policy for adding a byte order mark (BOM) to the output string. Supported keywords (default: none): <b>add</b> Add a BOM. <b>keep</b> Add a BOM if the input string has a BOM. <b>none</b> Don't add a BOM. <b>optimize</b> Add a BOM except if <i>outputformat=utf8</i> or <i>ebcdicutf8</i> and the output string contains only characters in the range < U+007F.
<b>errorpolicy</b>	(Keyword) Behavior in case of conversion errors (default: the value of the <i>errorpolicy</i> global option, see Table 2.1): <b>return</b> The replacement character will be used if a character reference cannot be resolved or a code or glyph ID doesn't exist in the specified font. An empty string will be returned in case of conversion errors. <b>exception</b> An exception will be thrown in case of conversion errors.
<b>font</b>	(Font handle; required for <i>inputformat=builtin</i> and <i>glyphid</i> ) Apply font-specific conversion according to the specified font.
<b>inflate</b>	(Boolean; only for <i>inputformat=utf8</i> ) If <i>true</i> , an invalid UTF-8 input string will not trigger an exception, but rather an inflated byte string in the specified output format will be generated. The inflated string contains Unicode characters which correspond to the ASCII interpretation of the bytes in the input string. This may be useful for debugging. Default: <i>false</i>
<b>output-format</b>	(Keyword) Unicode text format of the generated string: <i>utf8</i> , <i>ebcdicutf8</i> , <i>utf16</i> , <i>utf16le</i> , <i>utf16be</i> , <i>utf32</i> . An empty string is equivalent to <i>utf16</i> . Default: <i>utf16</i> Unicode-capable language bindings: the output format will be forced to <i>utf16</i> . C++ language binding: only the following output formats are allowed: <i>ebcdicutf8</i> , <i>utf8</i> , <i>utf16</i> , <i>utf32</i> .



## 2.3 Global Options

PDFlib offers various global options for controlling the library and the appearance of the PDF output. These options retain their settings across the life span of the PDFlib object, or until they are explicitly changed by the client.

---

```
C++ Java C# 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 2.3. The following options can be used:

- ▶ Options for resource handling and resource categories according to Table 2.3: *Encoding, enumeratefonts, FontAFM, FontnameAlias, FontOutline, FontPFM, HostFont, ICCProfile, resourcefile, saveresources, searchpath*
- ▶ Options for file handling and licensing according to Table 2.3: *avoiddemo stamp, filenamehandling, license, licensefile*
- ▶ Text filter options according to Table 2.3: *charref, escapesequence, glyphcheck, stringformat, textformat*
- ▶ Options for interactive elements according to Table 2.3: *hypertextencoding, hypertextformat, usehypertextencoding, usercoordinates*
- ▶ Other options according to Table 2.3: *asciifile, autospace, compress, kerning, logging, shutdownstrategy, usehostfonts, userlog*
- ▶ Option for error handling according to Table 2.1: *errorpolicy*
- ▶ Options for color handling according to Table 8.1: *iccprofilecmyk, iccprofilegray, iccprofilergb, preserveoldpantonenames, spotcolorlookup*

**Details** Except for resource category options new values override previously set option values.

The following options provide default values for the same-named text options (see Table 4.6 and Table 4.7):

*charref, escapesequence, glyphcheck, kerning, textformat*

At the same time these options change the options of the same name in the current text state. It is recommended to set options for content strings only in `PDF_set_text_option()` to avoid unwanted side effects.

**Scope** any, but restricted scopes apply to some options

Table 2.3 Global options for `PDF_set_option()`

option	description
<b>asciifile</b>	(Boolean; only supported on i5/iSeries and zSeries). Expect text files (PFA, AFM, UPR, encodings) in ASCII encoding. Default: true on i5/iSeries; false on zSeries
<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. This may be useful for generating Tagged PDF. Note that adding spaces changes the current text position. Default: false
<b>avoiddemo stamp</b>	(Boolean) 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

Table 2.3 Global options for `PDF_set_option()`

option	description
<b>charref</b>	(Boolean) If <code>true</code> , enable substitution of numeric and character entity references and glyph name references for all content, name and hypertext strings. In order to avoid character reference substitution in places where it is not desired (e.g. file names) it is recommended to set this option for content strings only in <code>PDF_set_text_option()</code> ; see PDFlib Tutorial for details. Default: <code>false</code>
<b>compress</b>	(Integer) Compression level from 0=no compression, 1=best speed, etc. to 9=best compression. This option does not affect image data handled in passthrough mode. Default: 6. Scope: any except object
<b>Encoding</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.
<b>enumerate-fonts</b>	<p>(Boolean) If <code>true</code>, PDFlib will search for font outline files in all directories which can be accessed via the SearchPath resource. This may take significant time if a large number of fonts is accessible, and should therefore be used with care. The generated resource list can be saved to a file with the <code>saveresources</code> option. The recommended strategy is to create and save the resource list only if the number of accessible fonts has changed, and not for each generated document or for each PDFlib object.</p> <p>For each valid font outline file PDFlib determines the font-family, font-weight and font-style names and synthesizes an API font name according to the following scheme:</p> <pre>&lt;font-family&gt;[, &lt;font-weight&gt;][, &lt;font-style&gt;]</pre> <p>PDFlib creates a FontOutline resource of the form <code>&lt;fontname&gt;=&lt;pathname&gt;</code> which connects the artificial font name with the full path name of the font. For PostScript Type 1 fonts the corresponding FontAFM or FontPFM resource will be created as well. In addition to the API font name PDFlib creates a Fontname-Alias resource with the PostScript name of the font if it is different from the artificial name:</p> <pre>&lt;PostScript fontname&gt;=&lt;artificial fontname&gt;</pre> <p>As a result, the font can be loaded via the artificial font name or its PostScript name. Default: <code>false</code></p>
<b>escape-sequence</b>	(Boolean) If <code>true</code> , enable substitution of escape sequences in all content, name and hypertext strings. In order to avoid escape sequence substitutions in places where it is not desired (e.g. file names) it is recommended to set this option for content strings only in <code>PDF_set_text_option()</code> ; Default: <code>false</code>
<b>filename-handling</b>	<p>(Keyword) Indicates the encoding of file names. File names supplied as function parameters without UTF-8 BOM in non-Unicode aware language bindings are interpreted according to this option to guard against characters which would be illegal in the file system and to create a Unicode version of the file name. An error occurs if the file name contains characters outside the specified encoding. Default: <code>unicode on Windows and OS X, auto on i5/iSeries, otherwise honorlang</code></p> <p><b>ascii</b> 7-bit ASCII</p> <p><b>basicebcdic</b> Basic EBCDIC according to code page 1047, but only Unicode values <code>&lt;= U+007E</code></p> <p><b>basicebcdic_37</b> Basic EBCDIC according to code page 0037, but only Unicode values <code>&lt;= U+007E</code></p> <p><b>honorlang</b> (Not on i5/iSeries) The environment variables <code>LC_ALL</code>, <code>LC_CTYPE</code> and <code>LANG</code> are interpreted The codeset specified in <code>LANG</code> is applied to file names if it is available.</p> <p><b>legacy</b> Use host encoding to interpret the file name</p> <p><b>unicode</b> Unicode encoding in (EBCDIC-) UTF-8 format</p> <p><b>all valid encoding names</b> Any encoding recognized by PDFlib (see Table 4.2) except CMaps, glyphid and builtin</p>
<b>FontAFM</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.
<b>Fontname-Alias</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.
<b>FontOutline</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.

Table 2.3 Global options for `PDF_set_option()`

option	description
<b>FontPFM</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.
<b>glyphcheck</b>	(Keyword) See Table 4.6 for a description. It is recommended to set this option for content strings only in <code>PDF_set_text_option()</code> ; see PDFlib Tutorial for details. Default: replace
<b>HostFont</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.
<b>hypertext-encoding</b>	(String; only for non-Unicode-capable language bindings) Encoding for hypertext strings. An empty string is equivalent to unicode. Default: auto
<b>hypertext-format</b>	(Keyword; only for non-Unicode-capable language bindings) Format for hypertext strings as function parameters. Supported keywords are bytes, utf8, ebcdicutf8, utf16, utf16le, utf16be, and auto. Default: auto
<b>ICCProfile</b>	(List of pairs of name strings) List of key/value pairs for a resource definition separated by whitespace or equal signs '=' (see PDFlib Tutorial for details). Multiple calls add new entries to the internal list.
<b>iccprofilecmyk</b> <b>iccprofilegray</b> <b>iccprofilergb</b>	(ICC profile handle) ICC profile which specifies a CMYK, Gray, or RGB color space for use with the icc-basedcmyk/gray/rgb color options. Default: no ICC color space
<b>kerning</b>	(Boolean) If true, enable kerning for fonts which have been opened with the readkerning option; disable kerning otherwise. Default: true
<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 <code>PDF_begin_document()</code> . 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 <code>PDF_begin_document()</code> .
<b>logging</b>	(Option list) Logging options according to Table 1.4
<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
<b>resourcefile</b>	(Name string) Relative or absolute file name of the PDFlib UPR resource file. The resource file will be loaded immediately before the first access to any resource. Existing resources will be kept; their values are overridden by new ones if they are set again.
<b>saveresources</b>	(Option list) Save the current resource list to a file. The following option is supported: <b>filename</b> The name of the resource file to which the resource list will be saved. Default: pdflib.upr
<b>searchpath</b>	(List of name strings) One or more relative or absolute path name(s) of directories 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). It is recommended to use double braces even for a single entry to avoid problems with directory names containing space characters. An empty string list (i.e. {} ) deletes all existing search path entries including the default entries. On Windows the search path can also be set via a registry entry. Default: platform-specific, see PDFlib Tutorial
<b>shutdown-strategy</b>	(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 PDFlib objects in a process; otherwise the behavior is undefined (default: 0): <ul style="list-style-type: none"> <li>0 A reference counter keeps track of how many PDFlib objects use the global resources. When the last PDFlib object is deleted the resources are released.</li> <li>1 The resources are kept until the end of the process. This may slightly improve performance, but requires more memory after the last PDFlib object is deleted.</li> </ul>

Table 2.3 Global options for `PDF_set_option()`

option	description
<b>stringformat</b>	(Keyword; only for non-Unicode-capable language bindings) Format used to interpret all strings at the API, i.e. name strings, content strings, hypertext strings and option lists. Supported keywords (default: legacy): <b>ebcdicutf8</b> (Only on i5/iSeries and zSeries) All strings and option lists are expected in EBCDIC-UTF-8 format with or without BOM. <b>legacy</b> Name strings, content strings, hypertext strings and option lists are treated according to the textformat, hypertextformat and hypertextencoding options. <b>utf8</b> (Not on i5/iSeries and zSeries) All strings and option lists are expected in UTF-8 format with or without BOM. The options textformat, hypertextformat and hypertextencoding are not allowed. The Textflow option fixedtextformat is forced to true. Legacy CJK CMaps can not be used for loading fonts. In the C language binding name strings as function parameters are still interpreted as UTF-16 strings if the length parameter is supplied with a value larger than 0. Use <code>PDF_convert_to_unicode()</code> to convert strings in 8-bit encodings to UTF-8.
<b>user-coordinates</b>	(Boolean) If false, coordinates for hypertext rectangles are expected in the default coordinate system; otherwise the current user coordinate system will be used. Default: false
<b>userlog</b>	String which will be copied to the log file
<b>usehostfonts</b>	(Boolean) If true, host fonts are included in the font search. Default: true
<b>usehypertext-encoding</b>	(Boolean; only for non-Unicode-capable language bindings) If true, the encoding specified in the hypertextencoding option will also be used for name strings. If false, the encoding for name strings without UTF-8 BOM is host. Default: false
<b>textformat</b>	(Keyword; only for non-Unicode capable language bindings) Format used to interpret content strings. Supported keywords: bytes, utf8, ebcdicutf8 (only on i5/iSeries and zSeries), utf16, utf16le, utf16be, and auto. Default: auto

C++ Java C# **double** `get_option(String keyword, String optlist)`

Perl PHP **float** `get_option(string keyword, string optlist)`

C **double** `PDF_get_option(PDF *p, const char *keyword, const char *optlist)`

Retrieve some option or other value.

**keyword** Keyword specifying the option to retrieve. The keywords below are supported; see description of `PDF_set_option()`, `PDF_set_text_option()` and `PDF_set_graphics_option()` regarding their meaning. Keywords for which no corresponding option exists are described in Table 2.4:

- ▶ Keywords for the string index of the *n*-th entry of the specified resource, where *n* corresponds to the *resourcenumber* option:  
*Encoding, FontAFM, FontnameAlias, FontOutline, FontPFM, HostFont, ICCProfile, searchpath*
- ▶ Keywords for Boolean option values return 1 for *true* or 0 for *false*:  
*asciifile, autospace, avoiddemonstamp, charref, decorationabove, escapesequenece, fakebold, kerning, overline, pdi, preserveoldpantonenames, spotcolorlookup, strikeout, tagged, topdown, underline, usercoordinates, usehostfonts, usehypertextencoding*
- ▶ Keywords for integer and float option values:  
*charspacing, compress, ctm\_a, ctm\_b, ctm\_c, ctm\_d, ctm\_e, ctm\_f, currentx, currenty, icccomponents, flatness, font, fontsize, horzscaling, iccprofilecmymk, iccprofilegray, iccprofilergb, italicangle, leading, linecap, linejoin, linewidth, major, minor, miterlimit, pageheight,*

*pagewidth, revision, scope, textrendering, textrise, textx, texty, underlineposition, underline-width, wordspacing*

- ▶ Keywords returning a string index for an option value or -1 if the string value is not available:  
*cliprule, errorpolicy, filenamehandling, fillrule, glyphcheck, hypertextencoding, hypertext-format, resourcefile, scope, textformat*
- ▶ Keywords for querying the current structure element (only in Tagged PDF mode):  
*activeitemid, activeitemindex, activeitemisinline, activeitemkidcount, activeitemname, activeitemstandardname*

Table 2.4 Additional keywords for `PDF_get_option()`

<b>keyword</b>	<b>description</b>
<b>activeitemid</b>	(Integer) Item id of the currently active structure item. This may be used with <code>PDF_activate_item()</code> or the parent suboption of <code>PDF_begin_item()</code> and the tag option. -1 is returned if no root element has been created yet. Scope: document, page
<b>activeitem-index</b>	(Integer) Zero-based index of the currently active structure item within its parent. This may be used with the index tag option. If the current item is a pseudo element or the root element or no root element has been created yet -1 is returned. Scope: document, page
<b>activeitem-isinline</b>	(Integer) 1 if the currently active structure item is an inline element, 0 otherwise. Scope: document, page
<b>activeitem-kidcount</b>	(Integer) Number of child elements of the currently active structure element created up to this point (not counting pseudo elements). -1 is returned if no root element has been created yet. Scope: document, page
<b>activeitem-name</b>	String index for the type name of the currently active structure element or pseudo element, or -1 if no root element has been created yet. Scope: document, page
<b>activeitem-standard-name</b>	String index for the standard element type name to which the currently active item is role mapped, or -1 if no root element has been created yet or the current item is a custom element for which no role mapping is available. If no rolemap is active the original type name is returned. Scope: document, page
<b>ctm_a</b> <b>ctm_b</b> <b>ctm_c</b> <b>ctm_d</b> <b>ctm_e</b> <b>ctm_f</b>	(Float) The components of the current transformation matrix (CTM) for vector graphics. Scope: page, pattern, template, glyph, path
<b>currentx</b> <b>currenty</b>	(Float) The x or y coordinate (in units of the current coordinate system), respectively, of the current point. Scope: page, pattern, template, glyph, path
<b>icccomponents</b>	(Integer) Number of color components in the ICC profile referenced by the handle provided in the iccprofile option
<b>major</b> <b>minor</b> <b>revision</b>	(Integer) Major, minor, or revision number of PDFlib, respectively. Scope: any, null <sup>1</sup>
<b>pageheight</b> <b>pagewidth</b>	(Float) Page size of the current page (dimensions of the MediaBox), template or glyph. Scope: any except object
<b>pdi</b>	(Integer) Returns 1 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 0. Scope: any, null <sup>1</sup>
<b>scope</b>	(Integer) String index for the name of the current scope (see Table 1.3)

Table 2.4 Additional keywords for `PDF_get_option()`

keyword	description
<code>textx</code> <code>texty</code>	(Float) The x or y coordinate of the current text position. Scope: page, pattern, template, glyph

1. C language binding: may be called with a PDF \* argument of NULL or 0

**optlist** Option list specifying an option according to Table 2.5.

**Returns** The value of some option as requested by *keyword*. If no value for the requested keyword is available, the function returns -1. If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with `PDF_get_string()`.

**Scope** any, but restricted scopes apply to some keywords

Table 2.5 Options for `PDF_get_option()`

option	description
<code>textstate</code>	(Boolean) If true, the values of the following options will be retrieved from the current text state, otherwise from the global options, (default: false): charref, escapesequences, glyphcheck, kerning, textformat
<code>iccprofile</code>	(ICC profile handle) ICC profile for use with the <code>iccomponents</code> keyword
<code>resource-number</code>	(Integer) Number of the resource to be retrieved; resources are numbered starting with 1. Default: 1

C++ Java C# **String get\_string(int idx, String optlist)**

Perl PHP **string get\_string(int idx, string optlist)**

C **const char \*PDF\_get\_string(PDF \*p, int idx, const char \*optlist)**

Retrieve a string value.

**idx** String index returned by one of the `PDF_get_option()` or `PDF_info_*` functions, or -1 if an option is supplied.

**optlist** An option list specifying options according to Table 2.6.

**Returns** The value of some string as requested by *idx* and *optlist*.

**Scope** Depends on the requested option.

**Bindings** C: The returned string is valid until the next call to any API function.

Table 2.6 Option for `PDF_get_string()`

option	description
<code>version</code>	(Boolean) Full PDFlib version string in the format <major>.<minor>.<revision>, possibly suffixed with additional qualifiers such as beta, rc, etc. Scope: any, null <sup>1</sup>

1. C language binding: may be called with a PDF \* argument of NULL or 0

---

C++ Java C# **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)**

---

Deprecated, use [PDF\\_set\\_option\(\)](#), [PDF\\_set\\_text\\_option\(\)](#), and [PDF\\_set\\_graphics\\_option\(\)](#).

---

C++ Java C# **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)**

---

Deprecated, use [PDF\\_set\\_option\(\)](#), [PDF\\_set\\_text\\_option\(\)](#), and [PDF\\_set\\_graphics\\_option\(\)](#).

---

C++ Java C# **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)**

---

Deprecated, use [PDF\\_get\\_option\(\)](#) and [PDF\\_get\\_string\(\)](#).

---

C++ Java C# **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)**

---

Deprecated, use [PDF\\_get\\_option\(\)](#).

---

## 2.4 Creating and Deleting PDFlib Objects

---

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 returns NULL or throws an exception.

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

**Bindings** C: In order to load the PDFlib DLL dynamically at runtime use *PDF\_new\_dl()*. *PDF\_new\_dl()* returns 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.

Other language bindings: 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 in C++.

**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.

*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 PDFlib object and frees all document-related PDFlib-internal resources. This function must only be called once for a given PDFlib 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*; no more API function calls with the same PDFlib object are allowed with the PDF object after this call.

*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.

## 2.5 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 C# 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). 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. In COM this is a variant. In REALbasic this is a MemoryBlock.

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

**optlist** An option list according to Table 2.7. The following option can be used: *copy*

**Details** The virtual file name can be supplied to any API function which uses input files. Use the *createpvf* option of *PDF\_begin\_document()* to create a PVF file which contains the generated PDF output. 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 2.7 Option for *PDF\_create\_pvf()*

option	description
<i>copy</i>	(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 C# **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 (see Table 2.3) The name of the virtual file as supplied to *PDF\_create\_pvf()*.

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

---

C++ Java C# **double info\_pvf(string filename, string keyword)**

Perl PHP **float info\_pvf(string filename, string keyword)**

C **double PDF\_info\_pvf(PDF \*p, const char \*filename, int len, const char \*keyword)**

---

Query properties of a virtual file or the PDFlib Virtual File system (PVF).

**filename** (Name string) The name of the virtual file. The filename may be empty if *keyword=filecount*.

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

**keyword** A keyword according to Table 2.7.

Table 2.8 Keywords for *PDF\_info\_pvf()*

<b>keyword</b>	<b>description</b>
<b>filecount</b>	Total number of files in the PDFlib Virtual File system maintained for the current PDFlib object. The filename parameter will be ignored.
<b>exists</b>	1 if the file exists in the PDFlib Virtual File system (and has not been deleted), otherwise 0
<b>size</b>	(Only for existing virtual files) Size of the specified virtual file in bytes.
<b>iscopy</b>	(Only for existing virtual files) 1 if the copy option was supplied when the specified virtual file was created, otherwise 0
<b>lockcount</b>	(Only for existing virtual files) Number of locks for the specified virtual file set internally by PDFlib functions. The file can only be deleted if the lock count is 0.

*Details* This function returns various properties of a virtual file or the PDFlib Virtual File system (PVF). The property is specified by *keyword*.

*Scope* any

## 2.6 PDF Object Creation API (POCA)

**Object types and frozen objects.** The PDF object creation API (POCA) is a low-level interface for creating PDF objects. POCA supports the following object types:

- ▶ simple object types: boolean, integer, name, float, string;
- ▶ container object types: array, dictionary, stream;
- ▶ specific types for PDFlib Blocks: percentage, color.

The generated PDF objects can be used as follows:

- ▶ with the *dpm* option of `PDF_begin/end_dpart()` to create document part metadata for PDF/VT;
- ▶ with the *blocks* option of `PDF_begin/end_page_ext()` to create PDFlib Blocks for use with PPS;
- ▶ with the *richmediaargs* option of `PDF_create_action()` to specify arguments for the ActionScript or JavaScript associated with a rich media annotation.

Supplying a PDF container object to any of the options listed above freezes the container object itself as well as all objects referenced from the container directly or indirectly, i.e. the full object tree created by the container will be frozen. Frozen objects can be used again with the options above, but they can no longer be modified with `PDF_poca_insert()` or `PDF_poca_remove()`.

---

```
C++ Java C# int poca_new(String optlist)
Perl PHP int poca_new(string optlist)
C int PDF_poca_new(PDF *p, const char *optlist)
```

---

Create a new PDF container object of type dictionary, array, or stream and insert objects.

**optlist** An option list for creating and populating a container.

- ▶ Options for creating a container according to Table 2.9: *containertype, usage*
- ▶ Options for inserting objects in the container according to Table 2.11: *direct, hypertextencoding, index, key, type, value, values*

**Returns** A POCA container handle which can be used until it is deleted with `PDF_poca_delete()`.

**Details** This function creates an empty PDF container object of the specified container type. The container can immediately be populated in the same call or later calls to `PDF_poca_insert()`.

**PDF/VT** A POCA container handle for an object of type dictionary with *usage=dpm* can be supplied as Document Part Metadata (DPM) with the *dpm* option of `PDF_begin/end_dpart()`.

**Scope** any

Table 2.9 Options for `PDF_poca_new()`

option	description
<b>container-type</b>	(Keyword; required) Type of the container: dict, array, or stream. Unspecified array slots and array slots which have been removed without inserting a new object will contain the keyword null in the PDF output. Note: containertype=stream is not yet implemented.

Table 2.9 Options for `PDF_poca_new()`

option	description
<b>usage</b>	(Keyword; required) Context in which the new container will be used. This option enables some checks to make sure that the container is suited for the intended use:
<b>blocks</b>	(Only relevant for <code>containertype=dict</code> ; only in the PPS product) The Block dictionary (the container which will be supplied to the <code>blocks</code> option of <code>PDF_begin/end_page_ext()</code> ) must contain one or more PDFlib Block definitions. The option <code>usage=blocks</code> must also be supplied to all container objects which will directly or indirectly be inserted into the new dictionary.
<b>dpm</b>	(Only relevant for <code>containertype=dict</code> ) All keys in the new dictionary or any dictionary contained in it must consist of ASCII characters, must conform to the rules of an XML NMTOKEN. This ensures that the dictionary can be used as Document Part Metadata (DPM) dictionary for PDF/VT. The option <code>usage=dpm</code> must also be supplied to all container objects which will directly or indirectly be inserted into the new dictionary.
<b>richmediaargs</b>	(Only for <code>containertype=array</code> ) The array can contain objects of type string, integer, float, or Boolean. However, the following is recommended to pass parameters from PDF to Flash: if a parameter for an ActionScript function parameter has type string, number, or int, use <code>type=string</code> in POCA (i.e. numbers must be wrapped within strings); if the parameter is declared as Boolean, use <code>type=boolean</code> in POCA (i.e. do not wrap boolean values as string). The POCA types integer and float should not be used since Acrobat does not pass them correctly to ActionScript.

C++ Java C# `void poca_delete(int container, String optlist)`

Perl PHP `poca_delete(int container, string optlist)`

C `void PDF_poca_delete(PDF *p, int container, const char *optlist)`

Delete a PDF container object.

**container** A valid POCA container handle retrieved with `PDF_poca_new()`.

**optlist** An option list according to Table 2.10. The following option can be used:  
*recursive*

**Details** The container will be deleted and can no longer be used. If the container is referenced from another dictionary or array all dictionary references to the deleted container are removed, and all array references to the deleted container are replaced with the `null` object. POCA container objects are not automatically deleted in `PDF_end_document()`.

**Scope** any; must always be paired with a matching `PDF_poca_new()` call.

Table 2.10 Options for `PDF_poca_delete()`

option	description
<b>recursive</b>	(Boolean) If <code>true</code> , the container object itself and all objects referenced from it will be deleted recursively. This may be useful as a shortcut for deleting a full object tree which is no longer needed. Default: <code>false</code>

C++ Java C# **void poca\_insert(int container, String optlist)**

Perl PHP **poca\_insert(int container, string optlist)**

C **void PDF\_poca\_insert(PDF \*p, int container, const char \*optlist)**

Insert a simple or container object in a PDF container object.

**container** A valid POCA container handle retrieved with `PDF_poca_new()`. Frozen containers (see »Object types and frozen objects«, page 37) are not allowed since they can no longer be modified.

**optlist** An option list according to Table 2.11. The following options can be used: *direct*, *hypertextencoding*, *index*, *key*, *type*, *value*, *values*

**Details** This function inserts an object in a container. The order in which objects are inserted in a container is not significant. Inserted containers may be populated after insertion; it is not required that inserted containers be complete at the time of insertion.

Inserting an object into a container must not create a loop of direct objects within the object graph. For example, a directly inserted dictionary cannot include a direct reference to its container. In order to create cyclic references use *direct=false* to create indirect objects which can reference arbitrary other objects.

**Scope** any

Table 2.11 Options for `PDF_poca_new()`, `PDF_poca_insert()` and `PDF_poca_remove()`

option	description
<b>direct</b> <sup>1</sup>	(Boolean; only for type=array and dict; ignored for other types) If true, the object will be inserted directly in the container; if false, an indirect PDF object will be created and a reference to the indirect PDF object will be inserted in the container. Indirect objects are useful to save space in the generated PDF if an object is used more than once. Default: true
<b>hypertext-encoding</b>	(Keyword) Specifies the encoding for the key, value, and values options. An empty string is equivalent to unicode. Default: value of the global hypertextencoding option
<b>index</b>	(Integer; only for containers with type=array; required for <code>PDF_poca_remove()</code> ) The zero-based index at which the value(s) will be inserted or deleted in the array. The value -1 can be used to insert the element as the new last item. The array will grow as necessary to include an element with the specified index. If the array already contains a value at the specified index it will be replaced with the new value. Default for <code>PDF_poca_new()</code> and <code>PDF_poca_insert()</code> : -1
<b>key</b>	(Hypertext string; only for containers with type=dict and stream; required for type=dict) The key under which the value will be inserted in the dictionary container or the dictionary associated with the stream container. The key must not include the leading '/' slash character. The key must conform to the conditions specified in the dictionary's usage option. If the dictionary already contains an entry with the same key it will be replaced with the new value. For type=stream the key must be different from Length and Filter.
<b>type</b> <sup>1</sup>	(Keyword; required except for stream containers without the key option) Type of the inserted object: array, boolean, dict, integer, name, float, stream, string, percentage, color The following types are not allowed if the container has been created with usage=dpm: name (use type=string instead), stream The following types are only allowed if the container has been created with usage=blocks: color, percentage

Table 2.11 Options for `PDF_poca_new()`, `PDF_poca_insert()` and `PDF_poca_remove()`

option	description
<b>value</b> <sup>1</sup>	<p>(Data type according to the <code>type</code> option; exactly one of the options <code>value</code> and <code>values</code> must be provided)                      The value of the inserted object, subject to the container type and the <code>type</code> option:                      For array and dictionary containers:                      If <code>type=boolean</code> the value must have option type <code>string</code>, and must contain one of the strings <code>true</code> or <code>false</code>.                      If <code>type=string</code> or <code>name</code> the value must have option type <code>Hypertext string</code>, and must contain the target directly. Values for <code>type=name</code> are limited to 127 bytes in UTF-8 representation, and must not include the leading <code>'/'</code> slash character.                      If <code>type=integer</code> the value must have option type <code>integer</code>, and must contain the target directly.                      If <code>type=float</code> the value must have option type <code>float</code> or <code>integer</code>, and must contain the target directly.                      If <code>type=array</code>, <code>dict</code>, or <code>stream</code> the value must have option type <code>POCA container handle</code> (i.e. created with <code>PDF_poca_new()</code>) and must specify the inserted container. The inserted object must have been created with the same usage option as the container.                      For <code>type=percentage</code> the value must have option type <code>number</code>. It will be interpreted as a percentage value and must include the percent sign (e.g. <code>50%</code>). It will be written as <code>Block data type percentage</code>                      For <code>type=color</code> the value must have option type <code>color</code> (see Table 1.2, page 14). It will be written as <code>Block data type color</code>. The following color space keywords are not allowed: <code>iccbased</code>, <code>iccbasedgray</code>, <code>iccbasedrgb</code>, <code>iccbasedcmyk</code>, <code>pattern</code>                      In order to pass arbitrary strings with this option the option list syntax described in »Unquoted string values in option lists«, page 9, may be useful.</p>
<b>values</b> <sup>1</sup>	<p>(List of one or more values according to the <code>type</code> option; only for containers with <code>type=array</code>; exactly one of the options <code>value</code> and <code>values</code> must be provided) One or more values of the same type which will be inserted in the array at the position specified by the <code>index</code> option. See option <code>value</code> regarding the conditions for specific types. If the specified list contains only a single element, the effect is equivalent to the <code>value</code> option. If the list contains more than one element, all elements in the list will be inserted in the array sequentially, possibly overriding existing elements. The array will grow as necessary to include all elements in the specified list.</p>

1. Only for `PDF_poca_new()` and `PDF_poca_insert()`

C++ Java C# **`void poca_remove(int container, String optlist)`**

Perl PHP **`poca_remove(int container, string optlist)`**

C **`void PDF_poca_remove(PDF *p, int container, const char *optlist)`**

Remove a simple or container object from a PDF container object.

**container** A valid POCA dictionary or array handle retrieved with `PDF_poca_new()`. Frozen containers (see »Object types and frozen objects«, page 37) are not allowed since they can no longer be modified.

**optlist** The following options of `PDF_poca_insert()` in Table 2.11 can be used:  
`hypertextencoding`, `index`, `key`

**Details** This function removes an object from a container of type array or dictionary. Nothing happens if the specified object doesn't exist in the container.

**Scope** any



# 3 Document and Page Functions

## 3.1 Document Functions

---

C++ Java C# *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, see Table 2.3) 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. On Windows it is OK to use UNC paths or mapped network drives.

**len** (C language binding only) Length of *filename* (in bytes). 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 2.1) and *hypertextencoding* (see Table 2.3)
- ▶ Document options according to Table 3.1. Some of these options can also be specified in *PDF\_end\_document()*; in this case they have precedence over identical options specified in *PDF\_begin\_document()*:  
*associatedfiles, attachments, autoxmp, destination, groups, labels, linearize, metadata, moddate, objectstreams, openmode, optimize, pagelayout, portfolio, search, uri, viewer-preferences*
- ▶ Options for PDF compatibility and standards according to Table 3.2:  
*compatibility, limitcheck, nodenamelist, pdfa, pdfua, pdfvt, pdfx, recordlevel, uses-transparency*
- ▶ Options for Tagged PDF according to Table 3.3:  
*checktags, lang, rolemap, structuretype, tag, tagged*
- ▶ Security options according to Table 3.4:  
*attachmentpassword, masterpassword, permissions, userpassword*
- ▶ Output processing options according to Table 3.5:  
*createoutput, createpvf, filemode, flush, inmemory, recordsize, removefragments, tempdirname, tempfilenames*

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

**PDF/VT** The following option is not allowed: *groups*.

**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** ASP: the *MapPath* facility should be used to construct full path names to be passed to this function.

C, C++, Java, JScript: take care of properly escaping the backslash path separator. For example, the following denotes a file on a network drive: `\\\\malik\\rp\\foo.pdf`. *PDF\_begin\_document\_callback()* is only available in C and C++.

---

C++ Java C# ***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 option: *hypertextencoding* (see Table 2.3)
- ▶ 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, associatedfiles, attachments, autoxmp, destination, destname, labels, metadata, moddate, openmode, pagelayout, portfolio, 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.

**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<sup>1</sup></b>	(Action list; not 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> JavaScript actions to be performed after printing/after saving/before closing/before printing/before saving the document.
<b>associated-files<sup>1</sup></b>	(List of asset handles; only for PDF 2.0 and PDF/A-3) Asset handles for associated files according to PDF/A-3. The files must have been loaded with <code>PDF_load_asset()</code> and <code>type=attachment</code> .
<b>attachments</b>	(List of option lists or list of asset handles; not for PDF/X-1a/3 and PDF/A-1; PDF/A-2: only PDF/A-1 and PDF/A-2 documents can be attached; PDF/A-3: not allowed, use <code>associatedfiles</code> instead) Specifies document-level file attachments which have been loaded with <code>PDF_load_asset()</code> and <code>type=attachment</code> . It is OK to supply file attachments both in <code>PDF_begin_document()</code> and <code>PDF_end_document()</code> . However, asset handles can only be supplied in <code>PDF_end_document()</code> . Supported suboptions: see Table 13.6
<b>autoxmp</b>	(Boolean; will be forced to <code>true</code> for PDF/X-3/4/5 and PDF/A) If <code>true</code> , PDFlib will create XMP document metadata from document info fields (see Section 14.2, »XMP Metadata«, page 257). Default: <code>false</code>
<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 12.10.
<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 <code>PDF_add_nameddest()</code> , and will be used as the document open action.
<b>groups<sup>2</sup></b>	(List of strings; not allowed in PDF/VT mode or if a document part hierarchy is created) 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.
<b>labels</b>	(List of option lists) A list containing one or more option lists according to Table 3.6 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 <code>style/prefix/start</code> must be unique within a document. Default: no page labels
<b>linearize</b>	(Boolean; only for <code>PDF_begin_document()</code> ) If <code>true</code> , the output document will be linearized. On z/OS this option cannot be combined with an empty filename. Default: <code>false</code>
<b>metadata</b>	(Option list) Supply XMP document metadata (see Section 14.2, »XMP Metadata«, page 257). Individual XMP properties may be overridden with document info fields supplied with <code>PDF_set_info()</code> . In PDF/A mode the supplied XMP metadata must conform to additional requirements (see PDFlib Tutorial).
<b>moddate</b>	(Boolean) If <code>true</code> , the <code>ModDate</code> (modification date) document info key will be created for compliance with some preflight tools. Default: <code>false</code>

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

option	description
<b>objectstreams<sup>2</sup></b>	<p>(List of keywords; PDF 1.5; will be forced to false if optimize or linearize is true) Generate compressed object streams which significantly reduce output file size (default: {other nodocinfo}):</p> <p><b>bookmarks</b> Compress bookmark objects.</p> <p><b>docinfo</b> Compress document info fields.</p> <p><b>dpartarrays</b> Compress dictionaries related to the document part hierarchy.</p> <p><b>dpartdicts</b> Compress arrays related to the document part hierarchy.</p> <p><b>fields</b> Compress form fields.</p> <p><b>names</b> Compress objects for named destinations.</p> <p><b>none</b> Don't generate any compressed object streams (except for categories which are explicitly enabled after this option).</p> <p><b>other</b> All categories which are not explicitly disabled after this keyword, plus other object types which don't have their own keyword.</p> <p><b>pages</b> Compress the objects comprising the page tree.</p> <p><b>poca</b> Compress all simple objects created with the POCA interface.</p> <p><b>tags</b> Compress marked content tags.</p> <p><b>xref</b> Generate a compressed xref stream. This category will automatically be enabled if at least one of the other categories is enabled.</p> <p>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.</p>
<b>openmode</b>	<p>(Keyword) Set the appearance when the document is opened. Default: bookmarks if the document contains any bookmarks, otherwise none.</p> <p><b>none</b> Open with no additional panel visible.</p> <p><b>bookmarks</b> Open with the bookmark panel visible.</p> <p><b>thumbnails</b> Open with the thumbnail panel visible.</p> <p><b>fullscreen</b> Open in fullscreen mode (does not work in the browser).</p> <p><b>layers</b> (PDF 1.5) Open with the layer panel visible.</p> <p><b>attachments</b> (PDF 1.6) Open with the attachments panel visible.</p>
<b>optimize<sup>2</sup></b>	<p>(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</p>
<b>pagelayout</b>	<p>(Keyword) The page layout to be used when the document is opened (default: default):</p> <p><b>default</b> The default setting of the Acrobat viewer.</p> <p><b>singlepage</b> Display one page at a time.</p> <p><b>onecolumn</b> Display the pages continuously in one column.</p> <p><b>twocolumnleft</b> Display the pages in two columns, odd pages on the left.</p> <p><b>twocolumnright</b> Display the pages in two columns, odd pages on the right</p> <p><b>twopageleft</b> (PDF 1.5) Display the pages two at a time, with odd-numbered pages on the left.</p> <p><b>twopageright</b> (PDF 1.5) Display the pages two at a time, with odd-numbered pages on the right.</p>
<b>portfolio<sup>1</sup></b>	<p>(Option list; PDF 1.7) Suboptions for creating a PDF portfolio according to Table 12.13</p>
<b>search</b>	<p>(Option list; not in ISO 32000-1) Instruct Acrobat to attach a search index when opening the document. The following suboptions are supported:</p> <p><b>filename</b> (Hypertext string; required) Name of a file containing a search index</p> <p><b>indextype</b> (Name string) Type of the index; must be PDX for Acrobat. Default: PDX</p>

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

option	description
<code>uri</code>	(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: no base URI
<code>viewer-preferences</code>	(Option list) Option list specifying various viewer preferences according to Table 3.7. Default: empty

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

Table 3.2 Options for PDF compatibility and standards in `PDF_begin_document()`

option	description
<code>compatibility</code>	(Keyword; will be ignored if one of the <code>pdfa</code> , <code>pdfua</code> , <code>pdfvt</code> or <code>pdfx</code> 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): <ul style="list-style-type: none"> <li><b>1.4</b> PDF 1.4 requires Acrobat 5 or above.</li> <li><b>1.5</b> PDF 1.5 requires Acrobat 6 or above.</li> <li><b>1.6</b> PDF 1.6 requires Acrobat 7 or above.</li> <li><b>1.7</b> PDF 1.7 is specified in ISO 32000-1 and requires Acrobat 8 or above.</li> <li><b>1.7ext3</b> PDF 1.7 extension level 3 requires Acrobat 9 or above.</li> <li><b>1.7ext8</b> PDF 1.7 extension level 8 requires Acrobat X or above.</li> <li><b>2.0</b> PDF 2.0 is specified in ISO 32000-2.</li> </ul>
<code>limitcheck</code>	If true, the limit for the number of indirect PDF objects (8,388,607) is enforced in PDF/A-1/2/3 and PDF/X-4/5 modes. Default: true
<code>nodenamelist</code>	(List of name strings; required for <code>pdfvt=PDF/VT-1</code> and <code>pdfvt=PDF/VT-2</code> ) Names for all levels of the document part hierarchy. All names must consist of ASCII characters and must conform to the rules of an XML NMTOKEN. The first string specifies the name for level 0 in the document part hierarchy.
<code>pdfa</code>	(Keyword) Set the PDF/A conformance level to one of the following (default: none): <ul style="list-style-type: none"> <li>PDF/A-1a:2005, PDF/A-1b:2005 (implies <code>compatibility=1.4</code>)</li> <li>PDF/A-2a, PDF/A-2b, PDF/A-2u (implies <code>compatibility=1.7</code>)</li> <li>PDF/A-3a, PDF/A-3b, PDF/A-3u (implies <code>compatibility=1.7</code>)</li> <li>none</li> </ul> <p>PDF/A1-a:2005, PDF/A-2a, and PDF/A-3a imply <code>tagged=true</code>. PDF/A can simultaneously conform to other standards as follows:</p> <ul style="list-style-type: none"> <li><code>pdfx=PDF/X-1a:2003</code>, <code>PDF/X-3:2003</code>, <code>PDF/X-4</code></li> <li><code>pdfvt=PDF/VT-1</code></li> <li><code>pdfua=PDF/UA-1</code></li> </ul> <p>If multiple options for PDF standards are specified the lowest compatibility value is used.</p>
<code>pdfua</code>	(Keyword) Set the PDF/UA conformance level to one of the following (default: none): <ul style="list-style-type: none"> <li><b>PDF/UA-1</b> Implies <code>compatibility=1.7</code> and <code>tagged=true</code>.</li> <li><b>none</b> No PDF/UA output</li> </ul>
<code>pdfvt</code>	(Keyword) Set the PDF/VT conformance level to one of the following (default: none): <ul style="list-style-type: none"> <li><b>PDF/VT-1</b> Implies <code>pdfx=PDF/X-4</code>; any other value for the <code>pdfx</code> option is an error.</li> <li><b>PDF/VT-2</b> The <code>pdfx</code> option must specify one of <code>PDF/X-4p</code>, <code>PDF/X-5g</code>, <code>PDF/X-5pg</code>; any other value for the <code>pdfx</code> option is an error.</li> <li><b>none</b> No PDF/VT output</li> </ul>

Table 3.2 Options for PDF compatibility and standards in `PDF_begin_document()`

option	description
<b>pdfx</b>	(Keyword) Set the PDF/X conformance level to one of the following (default: none): PDF/X-1a:2003 (implies compatibility=1.4) PDF/X-3:2003 (implies compatibility=1.4) PDF/X-4, PDF/X-4p <sup>1</sup> (implies compatibility=1.6) PDF/X-5g, PDF/X-5pg <sup>1</sup> (implies compatibility=1.6) none
<b>recordlevel</b>	(Non-negative integer; only relevant if a document part hierarchy is created) Zero-based level of the document part hierarchy which corresponds to recipient records.
<b>uses-transparency</b>	(Boolean; only for PDF/VT) If false, none of the pages in the generated document will contain any transparent objects. PDFlib will throw an exception if this assertion is violated. Setting this option to false is allowed only for documents without transparency, and facilitates generation of encapsulated XObjects for PDF/VT since all XObjects will unconditionally be marked as encapsulated. Default: true

1. The PDFlib Tutorial contains an important note about Acrobat problems with referenced ICC profiles.

Table 3.3 Options for Tagged PDF in `PDF_begin_document()`

option	description
<b>checktags</b>	(Keyword; must be strict in PDF/UA mode) Specifies whether the structure element nesting rules (see PDFlib Tutorial) are checked for elements created with <code>PDF_begin_item()</code> or the tag option of various functions. This option is only provided as a migration aid. It does not affect the tags in imported pages (see option <code>checktags</code> of <code>PDF_open_pdi_document()</code> ). Supported keywords (default: strict): <b>none</b> Tag nesting rules are not enforced. This setting may result in an invalid structure hierarchy and is therefore not recommended. <b>relaxed</b> Similar to strict except that a few rules are not enforced (see PDFlib Tutorial). <b>strict</b> If a tag violates the nesting rules an exception will be thrown.
<b>lang</b>	(String; recommended if tagged=true) Set the primary 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 structure items on all levels of the structure tree, but should be set initially for the document as a whole. PDF/UA: the natural language must be specified with this option or with the <code>lang</code> suboption of individual structure elements.
<b>rolemap</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 and pseudo element types are not allowed for the second entry in a sublist. Standard element type names also can be mapped to other standard element types in order to assign different semantics to existing element types. Indirect mappings are allowed, i.e. a custom type is mapped to another custom type which is then mapped to a standard type. Pairs with identical entries are silently ignored. See Section 14.3, »Tagged PDF«, page 258, regarding the use of custom element types in Tagged PDF. Custom element type names must not start with the reserved prefix <code>P1ib</code> . In PDF/UA it is not allowed to remap standard element types.

Table 3.3 Options for Tagged PDF in `PDF_begin_document()`

option	description
<b>structuretype</b>	(Keyword; only for PDF/UA) Type of document structure. Supported keywords (default: weak): <b>strong</b> The document is strongly structured, i.e. the structure tree reflects the document's logical organization. The only allowed structure type for headings is H, while H1, H2, etc. are not allowed. Each node in the structure tree contains at most one H tag plus one or more paragraph tags P. <b>weak</b> The document is weakly structured, i.e. the structure tree is only a few levels deep with all headings, paragraph etc. as immediate children. Logical structure may be expressed with heading tags H1, H2, etc., while H is not allowed. Headings may not have any descendants.
<b>tag</b>	(Option list) Tagging options according to Table 14.4. The specified structure element comprises the document structure root and will be closed automatically in <code>PDF_end_document()</code> . Only grouping elements are allowed for the tagname suboption.
<b>tagged</b>	(Boolean) 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 258). If PDF/A-1a:2005, PDF/A-2a, PDF/A-3a or PDF/UA-1 mode is active this option will automatically be set to true. Default: false

Table 3.4 Security options for `PDF_begin_document()`; not allowed for PDF/A and PDF/X

option	description
<b>attachment-password<sup>1</sup></b>	(String <sup>2</sup> ; PDF 1.6; will be ignored if <code>userpassword</code> or <code>masterpassword</code> are set; can not be combined with the <code>linearize</code> and <code>optimize</code> options; not for PDF/A and PDF/X) File attachments will be encrypted using the supplied string as password. The rest of the document will not be encrypted. On EBCDIC platforms the password is expected in ebcdic encoding or EBCDIC-UTF-8.
<b>master-password<sup>1</sup></b>	(String; required if <code>permissions</code> 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. On EBCDIC platforms the password is expected in ebcdic encoding or EBCDIC-UTF-8. Default: empty
<b>permissions</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> Acrobat will prevent high-resolution printing. If <code>noprint</code> 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> (Implies <code>nomodify</code> ) 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> (implies <code>nomodify</code> and <code>noannots</code> ) 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 <code>noaccessible</code> . <b>noaccessible</b> (Deprecated in PDF 2.0; not allowed in PDF/UA) Acrobat will prevent extracting text or graphics for accessibility (e.g. a screenreader). <b>plainmetadata</b> (PDF 1.5) Keep XMP document metadata unencrypted even in an encrypted document.
<b>user-password<sup>1</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. On EBCDIC platforms the password is expected in ebcdic encoding or EBCDIC-UTF-8. Default: empty

1. In order to pass arbitrary strings with this option the option list syntax described in »Unquoted string values in option lists«, page 9, may be useful.

2. Characters outside of Winansi encoding are only allowed in passwords for `compatibility=1.7ext3` or above

Table 3.5 Output processing options for `PDF_begin_document()`

option	description
<b>createoutput</b>	(Boolean) If false, the filename parameter is ignored and no output file or memory area is created. This option implies <code>compress=0</code> , <code>linearize=false</code> and <code>optimize=false</code> . Default: true
<b>createpvf</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 <code>PDF_end_document()</code> . In this case <code>PDF_get_buffer()</code> 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>filemode</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 <code>recordsize</code> must be consistent with the parameters specified in this option. Example string: <code>recfm=fb,lrcl=80,space=(cyl,(1,5))</code> . Default: empty, or <code>recfm=v</code> for unblocked output
<b>flush</b>	(Keyword; only for <code>PDF_begin_document_callback()</code> ) 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>inmemory</b>	(Boolean; not for <code>PDF_begin_document_callback()</code> ) If true and the linearize or optimize 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>recordsize</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>remove-fragments</b>	If true, a partial PDF output document which exists after an exception will be removed in <code>PDF_delete()</code> . Such PDF fragments are never usable as documents. This option has no effect if an empty filename has been specified, i.e. for in-memory PDF generation. Default: false
<b>tempdirname</b>	(String; not for <code>PDF_begin_document_callback()</code> ) Directory where temporary files for the linearize and optimize options will be created. If this option is missing, PDFlib will generate temporary files in the current directory. This option will be ignored if the <code>tempfilenames</code> option has been supplied. Default: not present
<b>temp-filenames</b>	(List of two strings; only for z/OS and USS) 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 filename parameter must not be empty. Default: not present



Table 3.6 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 groups option in <code>PDF_begin_document()</code> .
<b>hypertext-encoding</b>	(Keyword) Specifies the encoding for the prefix option. An empty string is equivalent to unicode. Default: value of the global <code>hypertextencoding</code> option.
<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.
<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. <ul style="list-style-type: none"> <li><b>none</b> no page number; labels will only consist of the prefix.</li> <li><b>D</b> decimal arabic numerals (1, 2, 3, ...)</li> <li><b>R</b> uppercase roman numerals (I, II, III, ...)</li> <li><b>r</b> lowercase roman numerals (i, ii, iii, ...)</li> <li><b>A</b> uppercase letters (A, B, C, ..., AA, BB, CC, ...)</li> <li><b>a</b> lowercase letters (a, b, c, ..., aa, bb, cc, ...)</li> </ul>

Table 3.7 Suboptions for the viewerpreferences option in `PDF_begin_document()` and `PDF_end_document()`

option	description
<b>centerwindow</b>	(Boolean) If true, 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; only true allowed in PDF/UA mode) Display the Title document info field in Acrobat's title bar (true) or the file name (false). Default: true for PDF/UA, otherwise 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
<b>hidemenubar<sup>1</sup></b>	(Boolean) Specifies whether to hide Acrobat's menu bar. Default: false
<b>hidetoolbar<sup>1</sup></b>	(Boolean) Specifies whether to hide Acrobat's tool bars. Default: false
<b>hide-windowui<sup>1</sup></b>	(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 printclip viewarea viewclip</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>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

<sup>1</sup> 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 writes PDF documents to a named disk file. Alternatively, PDF document data are 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 C# `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. The function returns a language-specific data type for binary data. The returned buffer must be used by the client before calling any other PDFlib function.

**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.

COM: Most COM clients will use a Variant type to hold the buffer contents. JavaScript with COM does not allow to retrieve the length of the returned variant array (but it does work with other languages and COM).

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

## 3.3 Page Functions

---

C++ Java C# **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 3.8. The PDFlib Tutorial lists applicable page size limits in Acrobat. See also Table 3.9 for more details (options *width* and *height*).

Table 3.8 Common standard page size dimensions in points<sup>1</sup>

format	width	height	format	width	height	format	width	height
a0	2380	3368	a4	595	842	letter	612	792
a1	1684	2380	a5	421	595	legal	612	1008
a2	1190	1684	a6	297	421	ledger	1224	792
a3	842	1190				11x17	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 with page options according to Table 3.9. These options have lower priority than identical options specified in *PDF\_end\_page\_ext()*:

*action, artbox, associatedfiles, bleedbox, blocks, cropbox, defaultcmyk, defaultgray, defaultrgb, 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 to their default values, and establishes a coordinate system according to the *topdown* option.

**PDF/VT** The following options are not allowed: *group, pagenumber*.

**Scope** *document*; this function starts *page* scope, and must always be paired with a matching *PDF\_end\_page\_ext()* call.

---

C++ Java C# **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 3.9. Options specified in *PDF\_end\_page\_ext()* have priority over identical options specified in *PDF\_begin\_page\_ext()*. The following options can be used:

*associatedfiles, action, artbox, bleedbox, blocks, cropbox, defaultcmyk, defaultgray, defaultrgb, duration, group, height, label, mediabox, metadata, rotate, taborder, transition, transparency-group, trimbox, userunit, viewports, width*

**Scope** *page*; this function terminates *page* scope, and must always be paired with a matching `PDF_begin_page_ext()` call. In Tagged PDF mode all inline and pseudo items must be closed before calling this function.

Table 3.9 Page options for `PDF_begin_page_ext()` and `PDF_end_page_ext()`

option	description
<b>action</b>	(Action list; not for PDF/A) 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>associatedfiles</b>	(List of asset handles; only for PDF 2.0 and PDF/A-3) Asset handles for associated files according to PDF/A-3. The files must have been loaded with <code>PDF_load_asset()</code> and <code>type=attachment</code> .
<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>blocks</b>	(POCA container handle; may be supplied to <code>PDF_begin_page_ext()</code> or <code>PDF_end_page_ext()</code> , but not to both functions for the same page; only available in PPS) Handle for a dictionary container created with <code>PDF_poca_new()</code> which contains PDFlib Block definitions for the PDFlib Personalization Server (PPS). The specified Blocks will be attached to the page. The dictionary must have been created with the option <code>usage=blocks</code> . Default: no Blocks
<b>defaultgray</b> <sup>1</sup> <b>defaultrgb</b> <sup>1</sup> <b>defaultcmyk</b> <sup>1</sup>	(ICC handle or keyword) Set a default gray, RGB, or CMYK color space for the page according to the supplied ICC profile handle. The option <code>defaultrgb</code> also supports the keyword <code>srgb</code> . The specified color space will be used to map device-dependent gray, RGB, or CMYK colors on the page (but not within templates on the page).
<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; not allowed in PDF/VT mode or if a document part hierarchy is created) 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 is true) 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 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</code> , <code>a1</code> , <code>a2</code> , <code>a3</code> , <code>a4</code> , <code>a5</code> , <code>a6</code> , <code>b5</code> , <code>letter</code> , <code>legal</code> , <code>ledger</code> , <code>11x17</code>
<b>label</b>	(Option list) An option list according to Table 3.6 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.
<b>mediabox</b>	(Rectangle; not allowed if the <code>topdown</code> option is true) Change the MediaBox for the current page. The coordinates are specified in the default coordinate system. By default, the MediaBox will be created by using the <code>width</code> and <code>height</code> parameters. The <code>mediabox</code> option overrides the <code>width</code> and <code>height</code> options and parameters.

Table 3.9 Page options for `PDF_begin_page_ext()` and `PDF_end_page_ext()`

<b>option</b>	<b>description</b>
<b>metadata</b>	(Option list) Metadata for the page (see Section 14.2, »XMP Metadata«, page 257)
<b>pagenumber<sup>1</sup></b>	(Integer; not allowed in PDF/VT mode or if a document part hierarchy is created) If this option is specified with a value <i>n</i> , the page will be inserted before the existing page <i>n</i> within the page group specified in the group option (or the document if the document doesn't use page groups). If this option is not specified the page will be inserted at the end of the group.
<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; only structure allowed in PDF/UA) Keyword specifying the tab order for form fields and annotations (Default: structure in Tagged PDF mode for PDF 1.5 and above, otherwise none): <ul style="list-style-type: none"> <li><b>column</b> Column by column from top to bottom, where columns are ordered as specified by the direction 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.</li> <li><b>row</b> Row by row starting at the topmost row, where the direction within a row is as specified by the direction suboption of the <code>viewerpreferences</code> option of <code>PDF_begin/end_document()</code>.</li> </ul>
<b>topdown<sup>1</sup></b>	(Boolean) If true, the origin of the coordinate system at the beginning of the page will be assumed in the top left corner of the page, and <i>y</i> coordinates will increase downwards; otherwise the default coordinate system will be used. Default: false
<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 full-screen mode as presentations if <code>openmode=fullscreen</code> (see Table 3.1). Default: replace <ul style="list-style-type: none"> <li><b>split</b> Two lines sweeping across the screen reveal the page</li> <li><b>blinds</b> Multiple lines sweeping across the screen reveal the page</li> <li><b>box</b> A box reveals the page</li> <li><b>wipe</b> A single line sweeping across the screen reveals the page</li> <li><b>dissolve</b> The old page dissolves to reveal the page</li> <li><b>glitter</b> The dissolve effect moves from one screen edge to another</li> <li><b>replace</b> The old page is simply replaced by the new page</li> <li><b>fly</b> (PDF 1.5) The new page flies into the old page.</li> <li><b>push</b> (PDF 1.5) The new page pushes the old page off the screen</li> <li><b>cover</b> (PDF 1.5) The new page slides on to the screen and covers the old page.</li> <li><b>uncover</b> (PDF 1.5) The old page slides off the screen and uncovers the new page.</li> <li><b>fade</b> (PDF 1.5) The new page gradually becomes visible through the old one.</li> </ul>

Table 3.9 Page options for `PDF_begin_page_ext()` and `PDF_end_page_ext()`

option	description
<b>transparency-group</b>	<p>(Option list or keyword; not for PDF/A-1 and PDF/X-1/3; restrictions apply to PDF/A-2/3 and PDF/X-4/5) Create a transparency group for the page. The following keywords are supported (default: auto) :</p> <p><b>auto</b> If transparent objects are present on the page itself or on an imported PDF page, graphics or template, the <code>transparencygroup</code> option is automatically created with a suitable color space; otherwise no transparency group is created.</p> <p><b>none</b> (Not allowed for PDF/A-2/3 without output intent if transparency is used on the page) Don't create any transparency group for the page.</p> <p>The following suboptions can be used to explicitly create a transparency group:</p> <p><b>colorspace</b> (Keyword or ICC profile handle; required for PDF/A-2/3 without output intent if transparency is used on the page) Blending color space of the transparency group (default: none) :</p> <p><b>DeviceCMYK</b> PDF/A-2/3 and PDF/X-4/5: only allowed with a CMYK output intent or if the <code>defaultcmyk</code> option has been supplied.</p> <p><b>DeviceGray</b> PDF/A-2/3 and PDF/X-4/5: only allowed with a gray or CMYK output intent or if the <code>defaultgray</code> option has been supplied.</p> <p><b>DeviceRGB</b> PDF/A-2/3 and PDF/X-4/5: only allowed with an RGB output intent or if the <code>defaultrgb</code> option has been supplied.</p> <p><b>none</b> (Not allowed for PDF/A-2/3 without output intent if transparency is used on the page) No color space is emitted for the transparency group.</p> <p><b>srgb</b> Keyword for selecting the sRGB color space</p> <p><b>isolated</b> (Boolean) Specifies whether the transparency group is isolated. Default: false</p> <p><b>knockout</b> (Boolean) Specifies whether the transparency group is a knockout group. Default: false</p>
<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 <code>mm</code> , <code>cm</code> , or <code>m</code> 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>	<p>(List of option lists; PDF 1.7ext3) Specifies one or more georeferenced areas (viewports) on the page; see Section 12.7, »Geospatial Features«, page 238, for details.</p> <p>Viewports allow different geospatial references (specified by the <code>georeference</code> option) to be used on different areas of the page, e.g. for multiple maps. The ordering of the option lists in the <code>viewports</code> list is relevant for overlapping viewports: the last viewport which contains a point will be used for that point.</p>
<b>width</b>	(Float or keyword; not allowed if the <code>topdown</code> option is true) See height option.

1. Only for `PDF_begin_page_ext()`

C++ Java C# **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, color, text and layer states of the current page are saved internally. The page 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. In Tagged PDF mode all inline and pseudo items must be closed before calling this function.

---

C++ Java C# `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 3.10. The following options can be used:  
*group*, *pagenumber*

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

In Tagged PDF mode it must be kept in mind that resuming a page does not restore any structure item. Instead, the item which is active when `PDF_resume_page()` is called will be the current item for subsequent page contents. It is recommended to use `PDF_activate_item()` to restore a specific structure element on the page as parent for subsequently generated contents.

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

Table 3.10 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 groups 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 group option (or in the document if the document doesn't use page groups) will be resumed. If this option is missing the last page in the group will be resumed.



## 3.4 Layers

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

---

```
C++ Java C# 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). If *len* = 0 a null-terminated string must be provided.

**optlist** An option list with layer settings:

- ▶ General options: *hypertextencoding* and *hypertextformat* (see Table 2.3)
- ▶ Layer control options according to Table 3.11:  
*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/A** PDF/A-1: this function must not be called.  
PDF/A-2/3: some options are restricted.

**PDF/X** PDF/X-1/2/3: this function must not be called.  
PDF/X-4/5: some options are restricted.

**PDF/UA** Some options are restricted.

**Scope** any except *object*

Table 3.11 Options for *PDF\_define\_layer()*

option	explanation
<b>creatorinfo</b>	(Option list; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) 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 is visible by default. Default: true
<b>initial-exportstate</b>	(Boolean; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) 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/A-2/3, PDF/X-4/5, and PDF/UA) The layer's recommended printing state. If true, Acrobat includes the layer when printing the document. Default: true

Table 3.11 Options for `PDF_define_layer()`

option	explanation
<b>initial-viewstate</b>	(Boolean; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) The layer's recommended viewing state. If true, Acrobat displays the layer when opening the document. Default: true
<b>intent</b>	(Keyword) Intended use of the graphics: View or Design. Default: View
<b>language</b>	(Option list; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) Specifies the language of the layer: <ul style="list-style-type: none"> <li><b>lang</b> (String; required) The language and possibly locale in the format described in Table 3.1 for the lang option</li> <li><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</li> </ul>
<b>onpanel</b>	(Boolean; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) 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/A-2/3, PDF/X-4/5, and PDF/UA) 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/A-2/3, PDF/X-4/5, and PDF/UA) Specifies whether the layer is intended for printing: <ul style="list-style-type: none"> <li><b>subtype</b> (Keyword) One of Trapping, PrintersMarks, or Watermark specifying the kind of content in the layer.</li> <li><b>printstate</b> (Boolean) If true, Acrobat will activate the layer contents upon printing.</li> </ul>
<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 used on a page if it has been supplied to <code>PDF_begin_layer()</code> at least once on that page. Default: false unless the layer is included in a non-default variant with <code>listmode=visiblepages</code>
<b>zoom</b>	(List of floats or percentages; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) 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 <code>maxzoom</code> can be used to specify the largest possible zoom factor.

C++ Java C# **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 3.12.

Table 3.12 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

Table 3.12 Dependency and relationship types for layers

<b>type</b>	<b>notes; options specific for this type</b>
<b>Lock</b>	(PDF 1.6) The layers specified in the group option are 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. Setting the parent to invisible automatically sets its children to invisible. A layer cannot belong to more than one parent layer. Options specific for this type: children, parent
<b>Radiobtn</b>	Specify a radio button relationship between the layers specified in the group option. This means that at most one layer in the group is visible at a time, which is particularly useful for multiple language layers. Option specific for this type: group
<b>Title</b>	The layer specified in the parent option does not control any page contents directly, but serves as a hierarchical separator for the layers specified in the children option. Options specific for this type: children, parent
<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>visiblelayers</code>

**optlist** An option list for layer dependencies:

- ▶ General option: *hypertextencoding* (see Table 2.3)
- ▶ Layer dependency options according to Table 3.13: *basestate*, *children*, *createorderlist*, *defaultvariant*, *depend*, *includelayers*, *invisiblelayers*, *group*, *visiblelayers*, *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 consist of 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 does not display individual layer names, but only the names of the layer variants. Layer variants are presented in Acrobat 9 only, and only for PDF/X documents. Acrobat X and above do not display layer variants. For this reason the use of layer variants is not recommended.

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/A** PDF/A-1: this function must not be called.  
 PDF/A-2/3: some options are restricted.

**PDF/X** PDF/X-1/2/3: this function must not be called.  
 PDF/X-4/5: some options are restricted.

Layer variants were required in the superseded standard PDF/X-4:2008, but direct layer control (without variants) is allowed in the successor PDF/X-4:2010 which is supported by PDFlib.

**PDF/UA** Some options are restricted.

**Scope** any except *object*; Layer relationships should be specified after all layers have been defined.

Table 3.13 Options for `PDF_set_layer_dependency()`

option	explanation
<b>basestate</b>	(Keyword; only for type=Variant; not for PDF/A-2/3, PDF/X-4/5, and PDF/UA) Specify the visibility of all layers which are not explicitly configured in the <code>visiblelayers</code> and <code>invisiblelayers</code> options. Supported keywords (default: on): <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 type=Parent and Title) One or more layer handles specifying the layers subordinate to the provided parent layer.
<b>createorderlist</b>	(Boolean; only for type=Variant and defaultvariant=true) If true, Acrobat will display the names of all layers. The value true has the following implications (default: true): <ul style="list-style-type: none"> <li>▶ Acrobat 9 displays layer variants (if present) in the Layers panel, but not layer names, and emits PDF/X-4 validation errors for documents with <code>createorderlist=true</code> since this is not allowed in PDF/X-4:2008.</li> <li>▶ Acrobat X and above display the layer names in its Layers panel, but not the layer variants, and successfully validates documents with <code>createorderlist=true</code> since this is allowed in PDF/X-4:2010.</li> </ul>
<b>defaultvariant</b>	(Boolean; only for type=Variant) 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 type=GroupAllOn, GroupAnyOn, GroupAllOff, and GroupAnyOff) The layer which is controlled by the layers specified in the group option.
<b>group</b>	(List of layer handles; only for type=GroupAllOn, GroupAnyOn, GroupAllOff, GroupAnyOff, Lock, and Radiobtn) One or more layer handles comprising the group. For type=Lock all layers in the group will be locked.
<b>includelayers</b>	(List of layer handles; only for type=Variant) 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 type=Variant) 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 type=Variant) Specify which layer names will be displayed in Acrobat's layer pane. Supported keywords (default: <code>visiblepages</code> ): <b>allpages</b> The names of all layers on all pages will be displayed. <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. In Acrobat this has an effect only if <code>defaultvariant=true</code> .
<b>parent</b>	(Layer handle; only for type=Parent and Title) The layer which is the parent of the layers specified in the <code>children</code> option.
<b>variantname</b>	(Hypertext string; required for type=Variant) Name of the selected variant. If type=Variant each variant name must be specified only once. Default if type is different from Variant: the default variant
<b>visiblelayers</b>	(List of layer handles; only for type=Variant) 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 C# **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, graphics, templates, and form fields can be controlled with the *layer* option of the respective functions.

**Scope** *page*

---

C++ Java C# **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

---

```
C++ Java C# 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 overrides this parameter. See option *fontname* in Table 4.2 for details.

**len** (C language binding only) Length of *fontname* in bytes. 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 overrides this parameter. See option *encoding* in Table 4.2 for details. Note the following 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 Table 2.1)
- ▶ Font loading options according to Table 4.2:  
*ascender, autosubsetting, capheight, descender, dropcorewidths, embedding, encoding, fallbackfonts, fontname, initialsubset, keepfont, keepnative, linegap, metadata, optimizeinvisible, preservepua, readfeatures, readkerning, readselectors, readshaping, replacement-char, simplefont, skipembedding, skipposttable, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*

**Returns** 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: 0) will be returned or an exception raised. The error behavior can be changed with the *errorpolicy* option.

If the function returns an error 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 4.1).

In non-Unicode language bindings, the option *textformat=auto* behaves as follows (note that all UTF formats are allowed for both cases):

- ▶ Wide character encodings: text in the loaded font is expected in text format *utf16* (for *encoding=glyphid* surrogates will not be interpreted)
- ▶ Byte- and multibyte encodings: text in the loaded font is expected in text format *bytes*.

*PDF/A* All fonts must be embedded.

*PDF/UA* All fonts must be embedded.

*PDF/X* All fonts must be embedded.

Table 4.1 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



Table 4.2 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) Deprecated and no longer functional due to internal changes in the font engine.
<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 for PDF/A, PDF/UA, and PDF/X) The widths for unembedded core fonts will not be emitted in the generated PDF. The 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, PDF/UA 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 embedding 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> <p>Font embedding can also be controlled with the <code>skipembedding</code> option.</p>
<b>encoding</b>	<p>(String; required for implicit font loading if the text appearance option <code>font</code> is not specified) Encoding in which incoming text for this font is interpreted:</p> <p>Wide character encodings:</p> <ul style="list-style-type: none"> <li>▶ unicode or the name of a Unicode CMaps</li> <li>▶ 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 <code>winansi</code>, <code>macroman</code>, <code>macroman_apple</code>, <code>ebcdic</code>, <code>ebcdic_37</code>, <code>pdfdoc</code>, <code>iso8859-X</code>, or <code>cpXXXX</code>, and non-Unicode (legacy) CMaps</li> <li>▶ (not for Unicode-capable language bindings) <code>cp932</code>, <code>cp936</code>, <code>cp949</code>, or <code>cp950</code> for CJK codepages</li> <li>▶ <code>host</code> or <code>auto</code> or the name of a user-defined encoding or an encoding known to the operating system</li> <li>▶ <code>builtin</code> to select the font's internal encoding (mostly for symbolic fonts);</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 <code>text</code> parameter is empty and the default <code>text</code> property is used, or the <code>font</code> option has been supplied.</p>

Table 4.2 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>fallbackfonts</b>	<p>(List of option lists according to Table 4.3) 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 4.1).</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 <code>font</code> 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><b>PDF_load_font():</b> the font name can alternatively be provided as function parameter.</p>
<b>fontstyle</b>	<p>(Keyword; deprecated) Controls the creation of artificial font styles. Possible keywords are <code>normal</code>, <code>bold</code>, <code>italic</code>, <code>bolditalic</code>. All text created with this font will be styled with the <code>fakebold</code> and/or <code>italicangle</code> text appearance options as appropriate. Unless another value of <code>italicangle</code> has been set, <code>-12</code> is used.</p> <p>If this option is applied to one of the core fonts, the appropriate bold, italic, or bolditalic font variant will be selected instead of faking the font style. If no such font is available (e.g. applying bold to Times-Bold), the option is ignored. Default: <code>normal</code></p>
<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 PDF output file size by creating identical subsets, which 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 <code>false</code> the font will be deleted automatically in <code>PDF_end_document()</code>. If <code>true</code> the font can also be used in subsequent documents until <code>PDF_close_font()</code> has been called. Default: <code>true</code> if <code>PDF_load_font()</code> is called in object scope, otherwise <code>false</code></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 <code>false</code> if <code>embedding=true</code>) If <code>false</code>, text in this font will be converted to CID values when creating PDF output. The text supplied to API functions 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 <code>true</code>, 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: <code>false</code> for TrueType fonts or <code>embedding=true</code>, and <code>true</code> otherwise.</p>
<b>linegap</b>	<p>(Integer between -2048 and 2048) See ascender above.</p>
<b>metadata</b>	<p>(Option list) Supply metadata for the font (see Section 14.2, »XMP Metadata«, page 257)</p>

Table 4.2 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>monospace</b>	(Integer between 1 and 2048; not for PDF/A and PDF/UA; deprecated) 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 should only be used for standard CJK fonts, and is not supported for core fonts; it will be ignored if the font is embedded. Default: absent (metrics from the font will be used)
<b>optimize-invisible</b>	(Boolean; not for PDF/X-1/2/3) If true, fonts which are exclusively used for invisible text (i.e. text-rendering=3) will not be embedded even if embedding=true. 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
<b>preservepua</b>	(Boolean) If true, characters which are mapped to a Unicode value in the Private Use Area (PUA) in the font retain their PUA value in the PDF output. This may be useful if the PUA characters carry locally defined semantics such as Japanese Gaiji/EUDC characters. If false, PUA characters are mapped to U+FFFD (Unicode replacement character) in the ToUnicode CMap in the PDF output. Default: false
<b>readfeatures</b>	(Boolean; only relevant for TrueType and OpenType fonts and encoding=unicode, glyphid, 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 features option (see Table 5.4). Setting this option to false may speed up font loading if OpenType features are not required. Default: true
<b>readkerning</b>	(Boolean) Controls whether or not kerning values will be read from the font. Actually applying the kerning values to text is controlled by the kerning text option (see Table 4.7). Setting this option to false may speed up font loading if kerning is not required. Default: true
<b>readselectors</b>	(Boolean; only relevant for TrueType and OpenType fonts) If true, the variation selectors will be read from the font if available. This is required for automatically substituting Ideographic Variation Sequences (IVS) within Unicode text. Default: true
<b>readshaping</b>	(Boolean; only relevant for TrueType and OpenType fonts and the encodings unicode and glyphid) 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 shaping option (see Table 5.4). Setting readfeatures to false can save memory if shaping is not required. Default: true
<b>replacementchar</b>	(Unichar or keyword; only relevant for glyphcheck=replace; ignored for fonts loaded with a non-Unicode CMap or glyphid 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. If the font doesn't contain any glyph for the specified Unicode character, the behavior of auto will be applied. U+0000 can be used to specify the font's »missing glyph« symbol. For symbolic fonts loaded with encoding=builtin the byte code must be supplied instead of the Unicode value. The following keywords can be used as an alternative to a Unicode character (default: auto): <ul style="list-style-type: none"> <li><b>auto</b> The first character from the following list for which a glyph is available in the font will be used as a replacement character: U+00A0 (NO-BREAK SPACE), U+0020 (SPACE), U+0000 (missing glyph symbol).</li> <li><b>drop</b> No output will be created for the character.</li> <li><b>error</b> An exception will be thrown if a typographically similar character is not available. This may be used to avoid unreadable text output.</li> </ul>
<b>simplefont</b>	(Boolean; only relevant for TrueType fonts with an 8-bit encoding) If this option is true and embedding=false, the font is emitted as simple font instead of as CID font. For some PDF viewers (particularly Acrobat) this is required for successful font substitution if the font is not installed on the viewing machine. However, the font may not be usable in form fields. Default: false

Table 4.2 Font loading options for `PDF_load_font()` and implicit font loading

option	description
<b>skip-embedding</b>	<p>(List of keywords; only relevant for <code>embedding=true</code>) Silently ignore font embedding problems (i.e. <code>embedding=true</code>, but the font cannot be embedded for some reason) if the font satisfies one or more of the conditions specified in the list. This may be useful in situations where font embedding is desired, but an unembedded font is preferable over an unusable font in cases where embedding is not possible. Supported keywords:</p> <p><b>latincore</b> The font is included in the set of 14 Latin core fonts (see PDFlib Tutorial for full list).</p> <p><b>standardcjk</b> The font is included in the set of standard CJK fonts (see PDFlib Tutorial for full list).</p> <p><b>fstype</b> The font is a TrueType or OpenType font and doesn't permit embedding according to the <code>fsType</code> flag in the font's OS/2 table.</p> <p><b>metricsonly</b> Only the PFM or AFM metrics file for the font is available, but the font outline (PFA, PFB) is missing.</p> <p>Default: empty list</p> <p>In PDF/A, PDF/X and PDF/UA only an empty list is allowed.</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; ignored for Type 3 fonts) Disable automatic font subsetting if the percentage of glyphs used in the document related to the total number of glyphs in the font exceeds the provided percentage. Default: 100%</p>
<b>subsetminsize</b>	<p>(Float; ignored for Type 3 fonts) Disable automatic font subsetting if the size of the original font file is less than the provided value in KB. Default: 50</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: subsetting is enabled automatically based on the <code>subsetlimit</code>/<code>subsetminsize</code> settings.</p>
<b>unicodemap</b>	<p>(Boolean; must not be set to <code>false</code> for PDF/A-1a/2a/2u/3a/3u and PDF/UA) Controls 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 @) If <code>true</code>, the font will be prepared for vertical writing mode.</p>
<b>xheight</b>	<p>(Integer between -2048 and 2048) See ascender above.</p>

C++ Java C# **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:

Table 4.3 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 4.2 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 <code>PDF_load_font()</code> 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 <code>embedding=false</code> and <code>keepnative=true</code> .				
<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 are always 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. One of the following keywords can be supplied: <table border="0" style="margin-left: 20px;"> <tr> <td><b>gaiji</b></td> <td>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.</td> </tr> <tr> <td><b>all</b></td> <td>All glyphs in the fallback font will be used to replace the corresponding characters in the base font, even if the character is available in the base font.</td> </tr> </table>	<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>all</b>	All glyphs in the fallback font will be used to replace the corresponding characters in the base font, even if the character is available in the base font.
<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>all</b>	All glyphs in the fallback font will be used to replace the corresponding characters in the base font, even if the character is available in the base font.				
<b>textrise</b>	(Float or percentage) Text rise value (see Table 4.7). 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				

- ▶ 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 C# **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 4.5. 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*, *hostfont*, *kerningpairs*, *numglyphs*, *shapingsupport*, *vertical*
- ▶ Keywords for Ideographic Variation Selectors: *maxuvsunicode*, *minuvsunicode*, *selector*, *selectorlist*
- ▶ Font/encoding relationship: *codepage*, *codepagelist*, *encoding*, *fallbackfont*, *keepnative*, *maxcode*, *numcids*, *numusableglyphs*, *predefcmap*, *replacementchar*, *symbolfont*, *unicodefnt*, *unmappedglyphs*
- ▶ Results of font processing for the current document: *numusedglyphs*, *usedglyph*, *willembed*, *willsubset*
- ▶ Color compatibility check for Type 3 fonts and PDF/A or PDF/X: *checkcolorspace*

**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 4.5.
- ▶ Mapping options according to Table 4.4 for specifying glyphs: *cid*, *code*, *glyphid*, *glyphname*, *selector*, *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 4.4 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>selector</i>	(Unichar) Unicode value of a variation selector in the range U+0xFE00..U+FE0F or U+E0100..U+E01EF. All values returned by the <i>selector</i> keyword can be supplied here.
<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 (in PHP: 0) will be returned. If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with `PDF_get_string()`.

**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

Table 4.5 Keywords and options for `PDF_info_font()`

keyword	explanation and options
<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> , <code>selector</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 encoding option was supplied and <code>font=-1</code> (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) 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 codepage).
<b>check-colorspace</b>	(Only relevant for Type 3 fonts with <code>colorized=true</code> ) 1 if the font can safely be used on the current page without risking a color-related violation of PDF/A or PDF/X; 0 otherwise
<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 true, the encoding name as specified in the API <b>actual</b> (Boolean) If true, 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 <code>unicode</code> 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: <code>unicode</code>
<b>familyname</b>	String index of the name of the font family, or -1 if unavailable

Table 4.5 Keywords and options for `PDF_info_font()`

keyword	explanation and options
<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> (Keyword; only if script is supplied) Specifies the language name. Default: <code>_none</code></p> <p><b>name</b> (Keyword; required) Specifies the four-character name of an OpenType feature table, e.g. <code>liga</code> (standard ligatures), <code>ital</code> (italic forms in CJK fonts), <code>vert</code> (vertical writing). Feature kern can not be queried.</p> <p><b>script</b> (Keyword) Specifies the script name. Default: <code>_none</code></p> <p>An exception is thrown if an unknown keyword for language, name, or script is supplied; see PDFlib Tutorial for lists of known keywords. The following return values indicate whether the specified OpenType feature table is present in the font and supported by PDFlib:</p> <p><code>-1</code> No feature tables are available in the font.</p> <p><code>0</code> The feature is not available for the specified script and language in the font, or is not known to PDFlib.</p> <p><code>1</code> The feature is available for the specified script and language.</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 <code>-1</code> if no feature tables are available.
<b>fontfile</b>	String index of the path name for the font outline file, or <code>-1</code> if unavailable
<b>fontname</b>	String index of the font name, or <code>-1</code> if unavailable. Supported options (default: <code>acrobat</code> ):
<b>api</b>	(Boolean) If <code>true</code> , the font name as specified in the API
<b>full</b>	(Boolean) If <code>true</code> , the <code>/FontName</code> entry in the PDF font descriptor
<b>acrobat</b>	(Boolean) If <code>true</code> , the font name as displayed in Acrobat
<b>fontstyle</b>	String index for the value of the fontstyle option (normal, bold, italic, or bolditalic)
<b>fonttype</b>	String index of the font type, or <code>-1</code> 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 <code>0...65535</code> specifying the font-internal id (GID) of the specified glyph, or <code>-1</code> if no such glyph was found. Supported options are the mapping options <code>cid</code> , <code>code</code> , <code>glyphid</code> , <code>glyphname</code> , <code>unicode</code> , <code>selector</code> .
<b>glyphname</b>	String index of the name of the specified glyph, or <code>-1</code> if no such glyph was found in the font or in the specified encoding (if the encoding option was supplied and <code>font=-1</code> (in PHP: <code>o</code> )). 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>hostfont</b>	<code>1</code> if the font is a host font, <code>0</code> otherwise
<b>italicangle</b>	Italic angle of the font (ItalicAngle in the PDF font descriptor)
<b>keepnative</b>	The resulting value of the <code>keepnative</code> option
<b>kerningpairs</b>	Number of kerning pairs in the font
<b>linegap</b>	Metrics value for the <code>linegap</code> . See <code>ascender</code> .
<b>maingid</b>	Glyph ID of the main glyph (member <code>mainGID</code> of <code>SING</code> table).
<b>maxcode</b>	Highest code value for the font's encoding, in particular: <code>0xFF</code> for single-byte encodings, <code>numglyphs-1</code> for <code>encoding=glyphid</code> , 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 <code>-1</code> if unavailable
<b>maxus-unicode</b>	Largest Unicode value which may be contained in a valid Ideographic Variation Sequence (IVS).



Table 4.5 Keywords and options for `PDF_info_font()`

<b>keyword</b>	<b>explanation and options</b>
<b>minuvs-unicode</b>	Smallest Unicode value which may be contained in a valid Ideographic Variation Sequence (IVS).
<b>monospace</b>	(Deprecated) 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
<b>numglyphs</b>	Number of glyphs in the font (including the .notdef glyph). Since GIDs start at 0 the highest possible GID value is one smaller than numglyphs.
<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. For TTC and WOFF fonts the keyword for the underlying base font format is returned, e.g. TTF. For CEF fonts the returned string is OTF.
<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 replacementchar option. For symbolic fonts loaded with encoding=builtin the code will be returned instead of the Unicode value.
<b>selector</b>	Unicode value of the variation selector with the number specified in the index option. If the index option is not specified or the specified selector is not available in the font, -1 is returned. Supported option: <b>index</b> (Non-negative Integer) Index of a selector.
<b>selectorlist</b>	String index of a string containing a space-separated list of the Unicode values of all variation selectors in the font. Each value is provided in the form hhhhh where h is a hexadecimal digit.
<b>shaping-support</b>	1 if the font supports shaping and the readshaping option was supplied when loading the font, otherwise 0
<b>singfont</b>	1 if the font is a SING (gaiji) font, otherwise 0
<b>standardfont</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 font=-1 (in PHP: 0)). Supported options are the mapping options cid, code, glyphid, glyphname, unicode, selector plus the following: <b>encoding</b> (String) Name of an 8-bit encoding
<b>unicodefont</b>	1 if the font/encoding combination provides Unicode mapping for the glyphs, otherwise 0. CJK fonts with non-Unicode CMaps and keepnative=true 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: glyphid
<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

Table 4.5 Keywords and options for `PDF_info_font()`

<b>keyword</b>	<b>explanation and options</b>
<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 xheight. See <code>ascender</code> .

## 4.2 Text Filter and Appearance Options

In this section the term *text* designates content strings, i.e. text with a specified appearance (font, color, etc.). In contrast, name strings and hypertext strings (e.g. file names) don't have any appearance; see PDFlib Tutorial for details.

Text options can be used with `PDF_set_text_option()`, `PDF_fit/info_textline()`, `PDF_fill_textblock()` and `PDF_add/create_textflow()`. Text options also apply to table cells and text Blocks. The following groups of text options are available:

- ▶ text filter options according to Table 4.6;
- ▶ text appearance options according to Table 4.7;
- ▶ shaping and typographic options according to Table 5.4 (not for `PDF_set_text_option()`).

Table 4.6 Text filter options for `PDF_set_text_option()`, `PDF_fit/info_textline()`, `PDF_fill_textblock()` and `PDF_add/create_textflow()`

option	explanation
<b>actualtext</b>	(Boolean; only for <code>PDF_set_text_option()</code> , <code>PDF_fit_textline()</code> , and <code>PDF_fill_textblock()</code> ) If true, PDFlib creates a marked content Span with suitable ActualText if the Unicode value(s) derived from the ToUnicode CMap wouldn't be correct and no user-provided Alt or ActualText was provided. This ensures correct text extraction for glyphs which are used to represent multiple similar looking Unicode values in a font, e.g. Ohm and Omega. Default: true
<b>charref</b>	(Boolean) If true, enable substitution of numeric and character entity references and glyph name references in content strings. <sup>1</sup> Default: the global charref option
<b>escape-sequence</b>	(Boolean) If true, enable substitution of escape sequences in content strings. <sup>1</sup> Default: the global escapesequences option
<b>glyphcheck</b>	(Keyword) Glyph checking policy: what happens if a code in the text cannot be mapped to a glyph in the selected font (default: the global glyphcheck option) <sup>1</sup> : <ul style="list-style-type: none"> <li><b>error</b> An exception will be thrown for unavailable glyphs. A detailed error message can be retrieved with <code>PDF_get_errmsg()</code>.</li> <li><b>none</b> No checking. notdef glyphs will trigger an exception in PDF/A, PDF/UA or PDF/X-4/5 mode; otherwise notdef glyphs may appear in the output.</li> <li><b>replace</b> Try to replace unavailable glyphs with typographically similar characters in the base and fallback fonts and decompose ligatures. If no suitable glyph could be found, the character will be handled according to the replacementchar option.</li> </ul>

Table 4.6 Text filter options for `PDF_set_text_option()`, `PDF_fit/info_textline()`, `PDF_fill_textblock()` and `PDF_add/create_textflow()`

option	explanation
<b>normalize</b>	(Keyword; ignored for encoding=glyhid and non-Unicode CMaps) Normalize incoming text to one of the Unicode normalization forms (default: none):
<b>none</b>	Do not apply any normalization. This is the default behavior; the client is responsible for supplying text which can be represented with glyphs from the selected font.
<b>nfc</b>	Normalization Form C (NFC): canonical decomposition followed by canonical composition. NFC replaces combining sequences with precomposed characters. This is useful for workflows with combining sequences since fonts usually contain only glyphs for the precomposed character. Without NFC normalization PDFlib emits a sequence of multiple characters instead of the precomposed character.
<b>nfkc</b>	Normalization Form KC (NFKC): compatibility decomposition followed by canonical composition. This is useful for workflows which are only interested in the semantics of characters, but not in formatting differences, e.g. convert shaped Arabic characters to their base form, resolve ligatures and fractions, replace vertical forms with horizontal forms, wide characters with regular characters.
<b>nfd</b>	Normalization Form D (NFD): canonical decomposition
<b>nfkd</b>	Normalization Form KD (NFKD): compatibility decomposition
	Since NFD and NFKD can create combining sequences they are unlikely to be useful in PDFlib workflows.
<b>textformat</b>	(Keyword; only for non-Unicode compatible language bindings) Format used to interpret content strings. Supported keywords: bytes, utf8, ebcdicutf8 (only on iSeries and zSeries), utf16, utf16le, utf16be, and auto. <sup>1</sup> Default: the global textformat option

1. The value may be overridden by a subsequent call to `PDF_set_option()` with the same option.






















Table 4.7 Text appearance options for `PDF_set_text_option()`, `PDF_fit/info_textline()`, `PDF_fill_textblock()` and `PDF_add/create_textflow()`

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 fontsize. In order to spread characters apart use positive values for horizontal writing mode, and negative values for vertical writing mode. Default: 0
<b>dasharray</b>	(List of two non-negative floats or keyword) The lengths of dashes and gaps for stroked (outline) text and decoration. The keyword none can be used to create solid lines. Default: none
<b>decoration-above</b>	(Boolean) If true, the text decoration enabled with the underline, strikeout, and overline options will be drawn above the text, otherwise below the text. Changing the drawing order affects visibility of the decoration lines, i.e. you can control whether the text overprints the lines or vice versa. Default: false
<b>fakebold</b>	(Boolean) If true, simulate bold text by stroking glyph outlines or multiple overprinting. Default: false
<b>fillcolor</b>	(Color) <sup>1</sup> Fill color of the text. Default for simple text output functions and <code>PDF_fit_textline()</code> with <code>inittextstate=false</code> : the corresponding option in the current graphics state. Default for <code>Textflow</code> and <code>PDF_fit_textline()</code> with <code>inittextstate=true</code> : {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 selected with <code>PDF_setfont()</code> for simple text output and <code>PDF_fit_textline()</code> with <code>inittextstate=false</code> . Otherwise no font is available which will trigger an error.

Table 4.7 Text appearance options for `PDF_set_text_option()`, `PDF_fit/info_textline()`, `PDF_fill_textblock()` and `PDF_add/create_textflow()`

option	explanation
<b>fontsize</b>	<p>(FontSize) Size of the font, measured in units of the current user coordinate system. In <code>PDF_fit_textline()</code> percentages relate to the box width (for <code>orientate=north</code> and <code>south</code>) or box height (for <code>orientate=east</code> and <code>west</code>). In <code>PDF_set_text_option()</code> and <code>Textflow</code> percentages relate to the size of the preceding text.</p> <p>Default: <code>PDF_setfont()</code> sets the default only for simple text output functions and <code>PDF_fit_textline()</code> with <code>inittextstate=false</code>. Otherwise no font size is available which will trigger an error.</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 created with this function. Default: no graphics state (i.e. current settings will be used).</p>
<b>horizscaling</b>	<p>(Float or percentage; must be different from 0) Horizontal text scaling to the given percentage. Text scaling shrinks or expands the text by a given percentage. Text scaling always relates to the horizontal coordinate. Default: 100%</p>
<b>inittextstate</b>	<p>(Boolean; only for <code>PDF_fit_textline()</code>) If <code>true</code> all text appearance options are initialized with the default values. If <code>false</code> the current text state values are used. Default: <code>false</code></p>
<b>italicangle</b>	<p>(Float; not supported for vertical writing mode) The italic (slant) angle of text in degrees (between <math>-90^\circ</math> and <math>90^\circ</math>). Negative values can be used to simulate italic text when only a plain upright font is available, especially for CJK fonts. Default: 0</p>
<b> Kerning</b>	<p>(Boolean) If <code>true</code>, enable kerning for fonts which have been opened with the <code>readkerning</code> option; disable kerning otherwise.<sup>2</sup> Default: the global option <code>kerning</code></p>
<b>leading</b>	<p>(Float or percentage) Specify the leading for multi-line text, i.e. the distance between baselines of adjacent lines of text as absolute value in user coordinates or as a percentage of font size. Setting the leading equal to the font size results in dense line spacing. However, ascenders and descenders of adjacent lines generally don't overlap (<code>leading=0</code> results in overprinting lines). Default: 100%</p> <p>The leading for <code>PDF_add/create_textflow()</code> is determined as follows: if there are option lists at the beginning of a line, the leading is 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 are only taken into account if <code>fixedleading=false</code>. If there are no option lists in the line, the previous leading value is used.</p>
<b>overline</b>	<p>(Boolean) If <code>true</code>, a line will be drawn above the text. Default: <code>false</code></p>
<b>shadow</b>	<p>(Option list; only for <code>PDF_fit_textline()</code>, <code>PDF_fill_textblock()</code>, <code>PDF_add/create_textflow()</code>) Create a shadow effect for the text (default: no shadow):</p> <ul style="list-style-type: none"> <li><b>disable</b> (Boolean; only for <code>PDF_add/create_textflow()</code>) If <code>true</code>, a previously specified shadow is disabled. Default: <code>false</code></li> <li><b>fillcolor</b> (Color) Fill color of the shadow. Default: <code>{gray 0.8}</code></li> <li><b>gstate</b> (Gstate handle) Graphics state created with <code>PDF_create_gstate()</code> which will be applied to the shadow. Default: <code>none</code></li> <li><b>offset</b> (List of 2 floats or percentages) The shadow's offset from the reference point of the text in user coordinates or as a percentage of the font size. Default: <code>{5% -5%}</code></li> <li><b>strokecolor</b> (Color; only effective if <code>textrendering</code> is set to <code>stroke text</code>) Stroke color of the shadow. Default: current stroke color</li> <li><b>strokewidth</b> (Float, percentage or keyword; only effective if <code>textrendering</code> is set to <code>stroke text</code>) Line width for outline text in the shadow (in user coordinates or as a percentage of the font size). The keyword <code>auto</code> or the equivalent value <code>0</code> uses a built-in default. Default: current stroke width if the main text is also set to <code>stroke text</code>, otherwise <code>auto</code></li> <li><b>textrendering</b> (Integer) Text rendering mode of the shadow. Default: current value of <code>textrendering</code></li> </ul>
<b>strikeout</b>	<p>(Boolean) If <code>true</code>, a line will be drawn through the text; see also <code>decorationabove</code>. Default: <code>false</code></p>
<b>strokecolor</b>	<p>(Color; only effective for stroked text, see <code>textrendering</code>) Stroke color of the text. Default: see <code>fillcolor</code></p>

Table 4.7 Text appearance options for `PDF_set_text_option()`, `PDF_fit/info_textline()`, `PDF_fill_textblock()` and `PDF_add/create_textflow()`

option	explanation		
<b>strokewidth</b>	(Float, percentage, or keyword; only effective if <code>textrendering</code> is set to stroke text) Line width for outline text (in user coordinates or as a percentage of the fontsize). The keyword <code>auto</code> or the equivalent value <code>o</code> uses a built-in default. Default: <code>auto</code>		
<b>tagtrailing-hyphen</b>	(Unichar or keyword; only relevant for Tagged PDF) If the last character in the text (after possibly applying glyph replacements) is equal to the specified Unicode value, it will be tagged as Span with Actual-Text soft hyphen <code>U+00AD</code> if required by the font, and no autospace will be added. The keyword <code>none</code> results in no tagging for soft hyphens. Default: <code>U+00AD</code>		
<b>textrendering</b>	(Integer) Text rendering mode. Only <code>textrendering=3</code> has an effect on Type 3 fonts (default: <code>o</code> ): <table border="0" style="width: 100%; margin-top: 10px;"> <tr> <td style="width: 50%; vertical-align: top;"> <p><b>o</b>  fill text</p> <p><b>1</b>  stroke text (outline)</p> <p><b>2</b>  fill and stroke text</p> <p><b>3</b> invisible text</p> </td> <td style="width: 50%; vertical-align: top;"> <p><b>4</b>  fill text and add it to the clipping path</p> <p><b>5</b>  stroke text and add it to the clipping path</p> <p><b>6</b>  fill and stroke text and add it to the clipping path</p> <p><b>7</b>  add text to the clipping path</p> </td> </tr> </table> <p>Behavior of <code>textrendering=4/5/6/7</code> (clipping modes):</p> <ul style="list-style-type: none"> <li>▶ There is no clipping effect after <code>PDF_fit_textflow()</code>, <code>PDF_fit_table()</code>, <code>PDF_fill_textblock()</code> and <code>PDF_fit_textline()</code> if the <code>textpath</code> option is specified.</li> <li>▶ Clipping areas can be accumulated across multiple calls to simple text output functions, but not across multiple calls to <code>PDF_fit_textline()</code>.</li> <li>▶ <code>PDF_fit_textline()</code>: the specified <code>fillcolor</code> and <code>strokecolor</code> remain in effect after the function call.</li> </ul>	<p><b>o</b>  fill text</p> <p><b>1</b>  stroke text (outline)</p> <p><b>2</b>  fill and stroke text</p> <p><b>3</b> invisible text</p>	<p><b>4</b>  fill text and add it to the clipping path</p> <p><b>5</b>  stroke text and add it to the clipping path</p> <p><b>6</b>  fill and stroke text and add it to the clipping path</p> <p><b>7</b>  add text to the clipping path</p>
<p><b>o</b>  fill text</p> <p><b>1</b>  stroke text (outline)</p> <p><b>2</b>  fill and stroke text</p> <p><b>3</b> invisible text</p>	<p><b>4</b>  fill text and add it to the clipping path</p> <p><b>5</b>  stroke text and add it to the clipping path</p> <p><b>6</b>  fill and stroke text and add it to the clipping path</p> <p><b>7</b>  add text to the clipping path</p>		
<b>textrise</b>	(Float or percentage) <code>Textrise</code> value, which specifies the distance between the desired text position and the baseline. Positive values of <code>textrise</code> move the text up. <code>Textrise</code> always relates to the vertical coordinate. This may be useful for superscripts and subscripts. Percentages are based on <code>fontsize</code> . Default: <code>o</code>		
<b>underline</b>	(Boolean) If <code>true</code> , a line will be drawn below the text. Default: <code>false</code>		
<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 <code>fontsize</code> ; a typical value is <code>-10%</code> ). The keyword <code>auto</code> specifies a font-specific value which will be retrieved from the font metrics or outline file. Default: <code>auto</code>		
<b>underline-width</b>	(Float, percentage, or keyword) Line width for underlined text (absolute value or percentage of the <code>fontsize</code> ). The keyword <code>auto</code> or the value <code>o</code> uses a font-specific value from the font metrics or outline file if available, otherwise <code>5%</code> . Default: <code>auto</code>		
<b>wordspacing</b>	(Float or percentage) <code>Wordspacing</code> , 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 ( <code>U+0020</code> ). The value is specified in user coordinates or a percentage of the <code>fontsize</code> . Default: <code>o</code>		

1. The value may be overridden by a subsequent call to `PDF_setcolor()` for simple text output functions and `PDF_fit_textline()` with `inittextstate=false`.
2. The value may be overridden by a subsequent call to `PDF_set_option()` with the same option.

---

C++ Java C# **void set\_text\_option(String optlist)**

Perl PHP **set\_text\_option(string optlist)**

C **void PDF\_set\_text\_option(PDF \*p, const char \*optlist)**

---

Set one or more text filter or text appearance options for simple text output functions.

**optlist** An option list specifying font and text options as follows:

- ▶ Text filter options according to Table 4.6:  
*actualtext, charref, escapesequence, glyphcheck, textformat*
- ▶ Text appearance options according to Table 4.7:  
*charspacing, dasharray, decorationabove, fakebold, fillcolor, font, fontsize, gstate, horzscaling, inittextstate, italicangle, kerning, leading, overline, strikeout, strokecolor, strokewidth, tagtrailinghyphen, textrendering, textrise, underline, underlineposition, underlinewidth, wordspacing*

**Details** The values of text options are relevant for all simple text output functions and *PDF\_fit\_textline()* with *inittextstate=false*. Calls to *PDF\_set\_text\_option()* should not be mixed with calls to *PDF\_setfont()* and *PDF\_setcolor()*.

All text options are reset to their default values at the beginning of a page, pattern, template, or glyph description, and retain their values until the end of the current *page*, *pattern*, *template*, or *glyph* scope. However, the text options can also be reset with the *inittextstate* option.

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

## 4.3 Simple Text Output

The functions listed in this section can be used for low-level text output. It is recommended to use the more powerful Textline and Textflow functions for more advanced text output (see Section 5.1, »Single-Line Text with Textlines«, page 89, and Section 5.2, »Multi-Line Text with Textflows«, page 95).

---

C++ Java C# **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 and font size to be used by simple text output functions (e.g. `PDF_show()` and `PDF_fit_textline()`). It is almost equivalent to a call to `PDF_set_text_option()` with the option list `font=<font> fontsize=<fontsize>`. However, unlike `PDF_set_text_option()` this function additionally sets the *leading* text option to *fontsize*.

The font must be set on each page before calling any of the simple text output functions. Font settings are not retained across pages.

The use of `PDF_set_text_option()` is recommended over `PDF_setfont()`.

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

---

C++ Java C# **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** New text position

**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*

---



---

C++ Java C# **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 bytes 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). If *len = 0* a null-terminated string must be provided.

**Details** The font and font size must have been set before with *PDF\_setfont()* or *PDF\_set\_text\_option()*. The current text position is moved to the end of the printed text.

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

**Bindings** *PDF\_show2()* is only available in C since in all other bindings arbitrary string contents can be supplied with *PDF\_show()*.

---

C++ Java C# **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 bytes 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). If *len = 0* a null-terminated string must be provided.

**Details** The font and font size must have been set before with *PDF\_setfont()* or *PDF\_set\_text\_option()*. The current text position is moved to the end of the printed text.

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

**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 C# **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 bytes when using `PDF_continue_text()`, since it is assumed to be null-terminated; use `PDF_continue_text2()` for strings which may contain null bytes.

**len** (Only for `PDF_continue_text2()`) Length of *text* (in bytes). 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* text option (which can be set with `PDF_set_text_option()`) 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.

**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 C# **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 bytes when using `PDF_stringwidth()`, since it is assumed to be null-terminated; use `PDF_stringwidth2()` for strings which may contain null bytes.

**len** (Only for `PDF_stringwidth2()`) Length of *text* (in bytes). 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.

**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 into account the values of the following text options (which can be set with `PDF_set_text_option()`): *horzscaling*, *kerning*, *charspacing*, and *wordspacing*.

---

*Scope* any except *object*

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

---

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

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

---

Deprecated, use `PDF_fit_textline()` with the `xadvancelist` option.

## 4.4 User-defined (Type 3) Fonts

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

```
C++ Java C# 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) 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) Option list according to Table 4.8. The following options can be used: *colorized*, *familyname*, *stretch*, *weight*, *widthsonly*

**Details** The font may contain an arbitrary number of glyphs. The font can be used until the end of the current *document* scope.

**Scope** any except *object*; 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 4.8 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) Font stretch value: ultracondensed, extracondensed, condensed, semicondensed, normal, semiexpanded, expanded, extraexpanded, ultraexpanded. Default: normal
<b>weight<sup>1</sup></b>	(Integer or keyword; PDF 1.5) Font weight: 100=thin, 200=extralight, 300=light, 400=normal, 500=medium, 600=semibold, 700=bold, 800=extrabold, 900=black. Default: normal
<b>widthsonly</b>	(Boolean) If true (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_ext()</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>widthsonly=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: false

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

---

C++ Java C# **void end\_font()**

Perl PHP **end\_font()**

C **void PDF\_end\_font(PDF \*p)**

---

Terminate a Type 3 font definition.

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

---

C++ Java C# **void begin\_glyph\_ext(int uv, String optlist)**

Perl PHP **begin\_glyph\_ext(int uv, string optlist)**

C **void PDF\_begin\_glyph\_ext(PDF \*p, int uv, const char \*optlist)**

---

Start a glyph definition for a Type 3 font.

**uv** Unicode value for the glyph. Each Unicode value can be supplied only for one glyph description. The glyph with the Unicode value 0 always gets glyph ID 0 and glyph name *.notdef*, regardless of whether or not the glyph was specified.

If *uv*=-1 the Unicode value is derived from the *glyphname* option according to PDFlib's internal glyph name list. If a glyph name is unknown, consecutive PUA values (starting at U+E000) will be assigned. This value can be queried with **PDF\_info\_font()**.

**optlist** Option list according to Table 4.9. The following options can be used: *boundingbox*, *code*, *glyphname*, *width*

*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 if the image has been opened with the *mask* option. This function resets all text, graphics, and color state parameters to their default values.

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

*Scope* *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\_ext()** and **PDF\_end\_glyph()** will be ignored.

Table 4.9 Options for **PDF\_begin\_glyph\_ext()**

option	description
<b>bounding-box</b>	(List of 4 floats; will be ignored in the second pass of the font definition for Type 3 font subsets and if the font's <i>colorized</i> option is <i>true</i> ) If the font's <i>colorized</i> option is <i>false</i> (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. Default: {0 0 0 0}
<b>code</b>	(Integer) Specify the glyph's slot number, i.e. its byte code in the Type 3 font's builtin encoding. By default the glyphs are numbered sequentially (starting with 0) in the order of creation.
<b>glyphname</b>	(String) Name of the glyph. The name for glyph 0 with Unicode 0 is forced to <i>.notdef</i> . Default: G<i> for glyph <i>=1,2,3,...
<b>width</b>	(Float; required; will be ignored in the second pass of the font definition for Type 3 font subsets) Width of the glyph in the glyph coordinate system as specified by the font's matrix.

---

C++ Java C# **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\_ext()* call.

---

C++ Java C# **void begin\_glyph(String glyphname, double wx, double llx, double lly, double urx, double ury)**

Perl PHP **begin\_glyph(string glyphname, float wx, float llx, float lly, float urx, float ury)**

C **void PDF\_begin\_glyph(PDF \*p,  
const char \*glyphname, double wx, double llx, double lly, double urx, double ury)**

---

Deprecated, use *PDF\_begin\_glyph\_ext()*.

## 4.5 User-defined 8-Bit Encodings

---

C++ Java C# `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





# 5 Text and Table Formatting

## 5.1 Single-Line Text with Textlines

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

---

C++ Java C# `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). 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 123, 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 Table 2.1)
- ▶ Font loading options according to Table 4.2 for implicit font loading (i.e. *font* option in the text appearance group not supplied):  
*ascender, autosubsetting, capheight, descender, embedding, encoding, fallbackfonts, fontname, fontstyle, keepnative, linegap, metadata, monospace, readfeatures, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, vertical, xheight*
- ▶ Text filter options according to Table 4.6:  
*actualtext, charref, escapesequence, glyphcheck, normalize, textformat*
- ▶ Text appearance options according to Table 4.7:  
*charspacing, dasharray, decorationabove, fakebold, fillcolor, font, fontsize, gstate, horizscaling, inittextstate, italicangle, kerning, leading, overline, shadow, strikeout, strokecolor, strokewidth, textrendering, textrise, underline, underlineposition, underline-width, wordspacing*
- ▶ Options for Textline formatting according to Table 5.1:  
*justifymethod, leader, textpath, xadvancelist*
- ▶ Shaping and typographic options according to Table 5.4:  
*features, language, script, shaping*
- ▶ Fitting options according to Table 6.1:  
*alignchar, blind, boxsize, fitmethod, margin, matchbox, orientate, position, rotate, stamp, showborder, shrinklimit*
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

**Details** If `inittextstate=false` (which is the default), the current text and graphics state options are 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 options are 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 are not modified by this function (in particular, the current font will be unaffected). However, the `textx/texty` options are adjusted to point to the end of the generated text output.

The reference point for `PDF_continue_text()` is not 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`

Table 5.1 Additional options for `PDF_fit_textline()`

option	explanation
<b>justifymethod</b>	(List of keywords; only relevant for <code>fitmethod=auto</code> and <code>stamp=none</code> ; requires <code>boxsize</code> ; ignored in vertical writing mode) Ensure that the text will not extend beyond the fitbox by applying one or more formatting methods without changing the fontsize. One or more of the following keywords can be supplied; if multiple keywords are present justification will be applied in the following order of decreasing priority: <code>wordspacing</code> , <code>charspacing</code> , <code>horizscaling</code> (default: none): <b>charspacing</b> Justify with an appropriate <code>charspacing</code> value. <b>horizscaling</b> Justify with an appropriate <code>horizscaling</code> value. <b>none</b> No justification <b>wordspacing</b> Justify with an appropriate <code>wordspacing</code> value. If the text does not contain any space characters <code>wordspacing</code> justification will not be applied.
<b>leader</b>	(Option list; ignored if <code>boxsize</code> 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 5.3 for a list of supported suboptions. Default: no leader
<b>textpath</b>	(Option list) Draw text along a path. Text beyond the end of the path will not be displayed. If <code>showborder=true</code> the flattened path will be drawn with the current linewidth and stroke color. The options listed in Table 5.2 plus the following options of <code>PDF_draw_path()</code> are supported: <b>align</b> , <b>attachmentpoint</b> , <b>boxsize</b> , <b>fitmethod</b> , <b>orientate</b> , <b>position</b> , <b>scale</b> (see Table 6.1) <b>close</b> , <b>round</b> , <b>subpaths</b> (see Table 7.7) <b>bboxexpand</b> , <b>boundingbox</b> (see Table 7.7) The following options of <code>PDF_fit_textline()</code> 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 ( <code>left/center/right</code> ). If the text is longer than the path it will always begin at <code>startoffset</code> . 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 ( <code>bottom/center/top</code> ). <b>rotate</b> Specifies a rotation angle for each glyph. The following fitbox-related options are ignored: <b>boxsize</b> , <b>margin</b> , <b>fitmethod</b> , <b>orientate</b> , <b>alignchar</b> , <b>showborder</b> , <b>stamp</b> , <b>leader</b> Kerning and text with CJK legacy encodings are not supported for text on a path.

Table 5.1 Additional options for `PDF_fit_textline()`

option	explanation
<b>xadvancelist</b>	(List of floats) Specifies the advance width of the glyphs in the text in user coordinates. The length of the list must be less or equal than the number of glyphs in the text. The xadvance values will be used instead of the standard glyph widths. Other effects, such as kerning and character spacing, are unaffected.

Table 5.2 Additional 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 <code>position</code> option is <code>top</code> the text will be placed on the other side 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> are used as reference point for the path.
<b>startoffset</b>	(Float or percentage) The offset of the starting point of the text along the 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%

Table 5.3 Suboptions for the `leader` option for `PDF_fit_textline()` and `PDF_add/create_textflow()` and inline options in `PDF_create_textflow()`

option	explanation
<b>font loading options</b>	If the font is specified implicitly (i.e. via the <code>fontname</code> and encoding options, as opposed to the <code>font</code> option), all font loading options according to Table 4.2 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):
<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).
<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.
<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.
<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.
<b>none</b>	No leader
<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.

Table 5.3 Suboptions for the leader option for `PDF_fit_textline()` and `PDF_add/create_textflow()` and inline options in `PDF_create_textflow()`

option	explanation
<code>fillcolor</code>	(Color) Color of the leader. Default: color of the text line
<code>font</code>	(Font handle) Handle for the font to be used for the leader. Default: font of the text line
<code>fontsize</code>	(FontSize) Size of the leader. Default: font size of the Textline
<code>text</code>	(Content string) The text which will be used for the leader. Default: U+002E ' (period)
<code>yposition</code>	(Float or keyword) Specifies the vertical position of the leader relative to the baseline as a numerical value or as one of the keywords <code>fontsize</code> , <code>ascender</code> , <code>xheight</code> , <code>baseline</code> , <code>descender</code> , <code>textrise</code> . Default: <code>baseline</code>

Table 5.4 Shaping and typographic options for `PDF_fit/info_textline()`, `PDF_add/create_textflow()`, and `PDF_fill_textblock()`

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

C++ Java C# **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:

- ▶ Keywords for querying the results of object fitting according to Table 6.3: *boundingbox, fitscalex, fitscaley, height, objectheight, objectwidth, width*
- ▶ Additional keywords according to Table 5.5: *angle, ascender, capheight, descender, endx, endy, pathlength, perpendiculardir, replacedchars, righttoleft, scalex, scaley, scriptlist, startx, starty, textwidth, textheight, unmappedchars, wellformed, writingdirx, writingdiry, xheight*

**optlist** An option list specifying options for *PDF\_fit\_textline()*. Options which are not relevant for the requested keyword are silently 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 returns 0 in case of an error. If *errorpolicy=exception* this function throws an exception in case of an error (even for the keyword *wellformed*).

**Scope** any except *object*

Table 5.5 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 user coordinates
<b>pathlength</b>	(Only for text on a path) Length of the path covered by the text from its starting point to the end point. This value can be queried even if <i>PDF_fit_textline()</i> was called in blind mode. The value can be used for the <i>startoffset</i> option of <i>PDF_fit_textline()</i> to continue labeling a path with additional text.
<b>perpendiculardir</b>	Unit vector perpendicular to <i>writingdir</i> ; 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 <i>glyphcheck=replace</i> .

Table 5.5 Keywords for `PDF_info_textline()`

<b>keyword</b>	<b>explanation</b>
<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>	Deprecated, use <code>fitscalex/fitscaley</code>
<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>textwidth, textheight</b>	Width and height of the text. The height is subject to the matchbox definition of <code>boxheight</code> , which defaults to <code>{capheight none}</code> .
<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 well-formed according to the selected font/encoding (and <code>textformat</code> , if applicable), otherwise 0.
<b>writingdirx writingdiry</b>	x/y coordinates of the dominant writing direction (i.e. the direction of inline text progression) which describes a unit vector from <code>(startx, starty)</code> to <code>(endx, endy)</code> . For left-to-right horizontal text the values will be <code>(1, 0)</code> , for vertical text <code>(0, -1)</code> , and for right-to-left horizontal text <code>(-1, 0)</code> . 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>	xheight in user coordinates

## 5.2 Multi-Line Text with Textflows

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

---

C++ Java C# **`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 as follows:

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ Font loading options according to Table 4.2 for implicit font loading (i.e. *font* option in the text appearance group not supplied):  
*ascender, autosubsetting, capheight, descender, embedding, encoding, fallbackfonts, fontname, fontstyle, keepnative, linegap, metadata, monospace, readfeatures, replacementchar, subsetlimit, subsetminsize, subsetting, unicodemap, xheight*
- ▶ Text filter options according to Table 4.6:  
*charref, escapesequence, glyphcheck, normalize, textformat*
- ▶ Text appearance options according to Table 4.7:  
*charspacing, dasharray, decorationabove, fakebold, fillcolor, font, fontsize, gstate, horzscaling, inittextstate, italicangle, kerning, leading, overline, shadow, strikeout, strokecolor, strokewidth, textrendering, textrise, underline, underlineposition, underline-width, wordspacing*
- ▶ Shaping and typographic options according to Table 5.4:  
*features, language, script, shaping*
- ▶ Options for Textflow formatting according to Table 5.6:  
*alignment, avoidemptybegin, fixedleading, hortabmethod, hortabsize, lastalignment, leader, leftindent, minlinecount, parindent, rightindent, ruler, tabalignment*
- ▶ Options for controlling the line break algorithm according to Table 5.7:  
*adjustmethod, advancedlinebreak, avoidbreak, locale, maxspacing, minspacing, nofitlimit, shrinklimit, spreadlimit*
- ▶ Command options according to Table 5.8:  
*comment, mark, matchbox, nextline, nextparagraph, restore, resetfont, return, save, space*
- ▶ Text semantics options according to Table 5.9:  
*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 (in PHP: 0), a new Textflow will be created and its handle will be returned. Otherwise the handle supplied in the *textflow* parameter will be re-

turned. By default, this function returns -1 (in PHP: 0) in case of an error. However, this behavior can be changed with the *errorpolicy* 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* (in PHP: 0) 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 preprocesses 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()* returns the string *\_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 5.6 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
	The value <i>alignment=justify</i> is ignored for a line containing the <i>nextline</i> option. The alignment of a line containing <i>nextparagraph</i> is not controlled by the <i>alignment</i> option, but rather the option <i>last-alignment</i> .



Table 5.6 Additional formatting options for `PDF_add/create_textflow()` and inline options in `PDF_create_textflow()`

option	explanation
<b>avoid-emptybegin</b>	(Boolean) If true, empty lines at the beginning of a fitbox will be deleted. Default: false
<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. <code>fixedleading</code> will be forced to true if the <code>wrap</code> option of <code>PDF_fit_textflow()</code> or the <code>createrwrapbox</code> suboption of the <code>matchbox</code> 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 is ignored (default: <code>relative</code> ): <b>relative</b> The position will be advanced by the amount specified in <code>hortabsize</code> . <b>typewriter</b> The position will be advanced to the next multiple of <code>hortabsize</code> . <b>ruler</b> The position will be advanced to the n-th tab value in the <code>ruler</code> 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 <code>relative</code> method will be applied.
<b>hortabsize</b>	(Float or percentage) Width of a horizontal tab <sup>1</sup> . The interpretation depends on the <code>hortabmethod</code> 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: <code>auto</code> ): <b>auto</b> Use the value of the alignment option if it is different from <code>justify</code> , else <code>left</code>
<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 5.3 for a list of supported suboptions. Default: no leader
<b>leftindent</b>	(Float or percentage) Left indent of text lines <sup>1</sup> . If <code>leftindent</code> 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 is added to <code>leftindent</code> . 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 <code>hortabmethod=ruler</code> <sup>1</sup> . The list may contain up to 32 non-negative entries in ascending order. Default: integer multiples of <code>hortabsize</code>
<b>tabalignment</b>	(List of keywords; only with <code>hortabmethod=ruler</code> ) 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 <code>ruler</code> option. Default: <code>left</code> . <b>center</b> Text will be centered at the tab position. <b>decimal</b> The first instance of <code>tabalignchar</code> will be left-aligned at the tab position. If no <code>tabalignchar</code> 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

Table 5.7 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 PDFlib 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 advanced line breaking algorithm which is required for complex scripts. This is required for line-breaking 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 <code>false</code> . 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 line-breaking 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 U+0020 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>
<b>spreadlimit</b>	(Float or percentage) Upper limit for the distance between characters for the <code>spread</code> method <sup>1</sup> ; the calculated distance will be added to the value of the <code>charspacing</code> option. Default: <code>0</code>

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

Table 5.8 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 <code>PDF_info_textflow()</code> and the <code>lastmark</code> 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 6.4
<b>nextline</b>	(Boolean) Force a new line; equivalent to one of the characters <code>U+000B</code> or <code>U+2028</code> . The options <code>alignment=justify</code> and <code>lastalignment</code> don't have any effect on the line containing the <code>nextline</code> option.
<b>nextparagraph</b>	(Boolean) Force a new paragraph; equivalent to one of the characters <code>U+000A</code> , <code>U+000D</code> , <code>U+000E</code> plus <code>U+000A</code> , <code>U+0085</code> , <code>U+2029</code> and <code>U+00FF</code> . The alignment of the line containing the <code>nextparagraph</code> option is determined by the option <code>lastalignment</code> ; the option <code>alignment</code> is ignored.
<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 first change of any font-related options which differ from the first setting, and will be ignored otherwise.
<b>restore</b>	(Boolean) If true, the values of all text and Textflow options saved by the most recent save command will be restored. A matchbox created within a save/restore pair will be retained after restore. Default: false
<b>return</b>	(String; must not start with an underscore <code>_</code> character) Exit <code>PDF_fit_textflow()</code> with the supplied string as return value.
<b>save</b>	(Boolean) If true, the values of all text and Textflow options will be saved, except those of the non-state options <code>nextline</code> , <code>nextparagraph</code> , <code>resetfont</code> , <code>return</code> , <code>space</code> , and <code>textlen</code> . Save/restore pairs can be nested to an arbitrary depth. Default: false
<b>space</b>	(Float or percentage) The text position will be advanced horizontally by the specified value <sup>1</sup> .

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

Table 5.9 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 <code>Uni-char</code> or a list of <code>Unichars</code> ; the <code>Unichars</code> must be <code>&lt; 0xFFFF</code> ; will be ignored if <code>advancedlinebreak=true</code> ) The specified <code>Unichars</code> will be classified by the specified keyword to determine the line breaking behavior of those character(s): <ul style="list-style-type: none"> <li><b>letter</b>     behave like a letter, e.g. <code>a B</code></li> <li><b>punct</b>     behave like a punctuation character, e.g. <code>+ / ; :</code></li> <li><b>open</b>      behave like an open parenthesis, e.g. <code>[</code></li> <li><b>close</b>     behave like a close parenthesis, e.g. <code>]</code></li> <li><b>default</b>    reset all character classes to PDFlib's builtin defaults</li> </ul> Example: <code>charclass={ close » open « letter {/ : =} punct &amp; }</code>

Table 5.9 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>tabalignchar</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 5.10. 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 5.10 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 (note that there must not be any equals character '=' between the macro name and its definition): <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 C# **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 100), and inline option lists according to Table 5.6 and Table 5.11 (see also »Inline option lists for Textflows«, page 102). 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 Table 2.1)
- ▶ All options of `PDF_add_textflow()` (see option list of `PDF_add_textflow()`)
- ▶ Options for controlling inline option list processing according to Table 5.11: *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* 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 102).

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 5.11 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 102). 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; only relevant for non-Unicode-capable language bindings and forced to <code>true</code> if <code>stringformat=utf8</code> ; this option doesn't make sense in inline option lists, and can only be used in the <code>optlist</code> parameter) If <code>true</code> , 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 <code>false</code> , 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: <code>false</code>
<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-capable languages) characters before the next inline option list (see »Inline option lists for Textflows«, page 102). 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 5.6. 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>
```

- ▶ Specify the *begoptlistchar* as an escape sequence and set the *escapesequence* global option to *true*. However, escape sequences don't work within inline option lists including the *endoptlistchar*.

*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 C# 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. The following options can be used:

- ▶ Textflow options according to Table 5.12: *avoidwordsplitting*, *blind*, *createfitttext*, *createlastindent*, *exchangeffillcolors*, *exchange-strokecolors*, *firstlinedist*, *fitmethod*, *fontscale*, *lastlinedist*<sup>1</sup>, *linespreadlimit*, *maxlines*, *minfontsize*, *orientate*, *returnatmark*, *rewind*, *rotate*, *showborder*, *showtabs*, *stamp*, *truncatetrailingwhitespace*, *verticalalign*<sup>1</sup>, *wrap*
- ▶ Matchbox option according to Table 6.1: *matchbox*

- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

**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 4.7) in *PDF\_create\_textflow()* or *PDF\_add\_textflow()* to control the appearance of the text. After returning from this function the text state is unchanged. However, the *textx/texty* options are 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 5.12 Options for *PDF\_fit\_textflow()*

option	explanation
<b>avoidword-splitting</b>	(Boolean) If <i>true</i> and <i>fitmethod=auto</i> , <i>Textflow</i> attempts to fit the text completely into the fitbox by decreasing the fontsize and avoiding word splitting (see <i>adjustmethod</i> ).
<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>fitttext</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: <ul style="list-style-type: none"> <li><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></li> <li><b>matchbox</b> (Option list according to Table 6.4) 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.</li> </ul>



Table 5.12 Options for `PDF_fit_textflow()`

option	explanation
<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 <code>Textflow</code> 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: <code>exchangefillcolors={{gray 0} white Orchid DeepPink {rgb 1 0 1} MediumBlue}</code>
<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 <code>Textflow</code> 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.
<b>firstlinedist<sup>1</sup></b>	(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> ): <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. <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. <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. <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. 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.
<b>fitmethod</b>	(Keyword) Specifies the method used to fit the text into the fitbox (default: <code>clip</code> ): <b>auto</b> <code>PDF_fit_textflow()</code> 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> ) <b>clip</b> The text will be truncated at the bottom of the fitbox. <b>nofit</b> The text can extend beyond the bottom of the fitbox.
<b>fontscale</b>	(Positive 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> .
<b>gstate</b>	(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)
<b>lastlinedist<sup>1</sup></b>	(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: <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. If <code>fixedleading=false</code> the maximum of all descender values found in the last line is used.
<b>linespread-limit</b>	(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%
<b>maxlines</b>	(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>

Table 5.12 Options for `PDF_fit_textflow()`

<b>option</b>	<b>explanation</b>
<b>minfontsize</b>	(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%
<b>mingapwidth</b>	(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 fontsize. This may be useful to avoid ugly formatting results in cases where only small gaps are left between wrap contours. Default: 10%
<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) <code>PDF_fit_textflow()</code> will return prematurely at the text position where the <code>Textflow</code> option <code>mark</code> is defined with the specified number. The return reason string will be <code>_mark#</code> , where <code>#</code> is the number specified in this option.
<b>rewind</b>	(Integer: -2, -1, 0, or 1) State of the supplied <code>Textflow</code> is reset to the state before some other call to <code>PDF_fit_textflow()</code> with the same <code>Textflow</code> handle (default: 0): <b>1</b> Rewind to the state before the first call to <code>PDF_fit_textflow()</code> . <b>0</b> Don't reset the <code>Textflow</code> . <b>-1</b> Rewind to the state before the last call to <code>PDF_fit_textflow()</code> . <b>-2</b> Rewind to the state before the second last call to <code>PDF_fit_textflow()</code> .
<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 <code>true</code> , the border of the fitbox and all wrap boxes is stroked (using the current graphics state). This may be useful for development and debugging. Default: <code>false</code>
<b>showtabs</b>	(Keyword) Tab stops and left indents are visualized with vertical lines as a debugging aid. The lines will be drawn according to the graphics state which was active before calling <code>PDF_fit_textflow()</code> (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 <code>newline</code> 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 <code>Textflow</code> . If this option is <code>true</code> , trailing whitespace is removed, i.e. the fitbox is treated as empty and the return value is <code>_stop</code> . If this option is <code>false</code> , the whitespace is processed like regular text, i.e. the function may return a value different from <code>_stop</code> (depending on the amount of trailing whitespace) and the <code>textendx/y</code> and other keywords of <code>PDF_info_textflow()</code> take the whitespace into account. <code>truncatetrailingwhitespace=false</code> may be useful if the original text must be processed without any whitespace removal. Default: <code>true</code>

Table 5.12 Options for `PDF_fit_textflow()`

option	explanation
<b>verticalalign</b> <sup>1</sup>	(Keyword) Vertical alignment of the text in the fitbox; the <code>firstlinedist</code> and <code>lastlinedist</code> options will be taken into account as appropriate (default: <code>top</code> ): <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 <code>linespreadlimit</code>. If this limit is exceeded no justification will be performed. The height of the first line will only be increased if <code>firstlinedist=leading</code>.</li> </ul>
<b>wrap</b>	(Option list according to Table 5.13) The text will run around the areas specified with the suboptions listed in Table 5.13. This can be used to place images or paths within the <code>Textflow</code> and wrap the text around it, or to fill arbitrary shapes with text. The fitbox will be filled according to the <code>fillrule</code> 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 <code>addfitbox</code> and <code>inversefill</code> 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 <code>llx/lly/urx/ury</code> 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: <code>wrap={ boxes={{120r 340r 50r 60r}} }</code>            (equivalent to <code>wrap={ boxes={{120 340 170 400}} }</code>)</p> <p>Exclude the upper right quarter of the fitbox: <code>wrap={ boxes={{50% 50% 100% 100%}} }</code></p> <p>Fill a triangular shape: <code>wrap={ addfitbox polygons={{50% 80% 30% 40% 70% 40% 50% 80%}} }</code></p> <p>Exclude the area of an image with a matchbox called <code>image1</code>: <code>wrap={ usematchboxes={{ image1 }}} }</code></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.

Table 5.13 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) 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>creatematch-boxes</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 6.4 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 7.1 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>usematch-boxes</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>repoint</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 <code>PDF_draw_path()</code> can also be used (see Table 6.1 and Table 7.7): 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.

---

C++ Java C# **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 5.14.

**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* options). If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with *PDF\_get\_string*().

**Scope** any except *object*

Table 5.14 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). <code>firstlinedist</code> and <code>lastlinedist</code> will be taken into account.
<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>fittext</b>	String index for 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 (EBDIC-)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>	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 <i>x</i> and <i>y</i> 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 <i>y</i> 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 <i>y</i> coordinate of the baseline of the shortest text line in the most recently filled fitbox, in current user coordinates
<b>returnreason</b>	String index for 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 <i>x</i> and <i>y</i> 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: 0           No word had to be split. 1           At least one word had to be split.
<b>textendx, textendy</b>	The <i>x</i> or <i>y</i> 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 5.14 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 in current user coordinates. <code>firstlinedist</code> and <code>lastlinedist</code> will be taken into account.

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

---

C++ Java C# **`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

## 5.3 Table Formatting

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

---

C++ Java C# `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). If *len* = 0 a null-terminated string must be provided.

**optlist** An option list specifying table cell formatting details:

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ Column and row definition options according to Table 5.15.:  
*colwidth, colscalegroup, minrowheight, return, rowheight, rowjoingroup, rowscalegroup*
- ▶ Cell property options according to Table 5.15.:  
*avoidwordsplitting, colspan, margin, marginleft, marginbottom, marginright, margintop, rowspan*
- ▶ Cell content formatting options according to Table 5.15.:  
*continuetextflow, repeatcontent*
- ▶ Static cell contents according to Table 5.16:  
*fitgraphics, fitimage, fitpath, fitpdipage, fittextflow, fittextline, graphics, image, matchbox, path, pdipage, textflow*
- ▶ Interactive cell contents according to Table 5.17 (only in *page* scope):  
*annotationtype, fieldname, fieldtype, fitannotation, fitfield*
- ▶ Option for abbreviated structure element tagging according to Table 14.5: *tag*

**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, graphics, 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.

PDF/UA Vector graphics and raster images supplied with the *path* or *image* options must be tagged as *Artifact* or *Figure* with the *tag* option.

Scope any except *object*

Table 5.15 Formatting options for `PDF_add_table_cell()`

key	explanation
<b>avoidword-splitting</b>	(Boolean; only relevant for <i>Textflow</i> cells) If <code>true</code> , the table formatter will check whether the <i>Textflow</i> 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: <code>true</code>
<b>checkword-splitting</b>	Deprecated, use <code>avoidwordsplitting</code>
<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 <code>column</code> 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 <code>PDF_fit_table()</code> ). 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, graphics and PDF pages in table cells don't have any influence on column widths. Default: see option <code>colwidthdefault</code> of <code>PDF_fit_table()</code>
<b>continue-textflow</b>	(Boolean; only relevant for <i>Textflows</i> ) If <code>true</code> the contents of the <i>Textflow</i> specified in the <code>textflow</code> option can be continued in another cell provided that the other cell is filled with the same <i>Textflow</i> handle and <code>continuetextflow=true</code> as well. The parts of the <i>Textflow</i> will be placed in the order in which the cells are added. PDFlib will not adjust the cell size to the whole <i>Textflow</i> , and the <code>avoidwordsplitting</code> option will be ignored. Therefore, a suitable cell size should be defined. If <code>false</code> the <i>Textflow</i> will be started from the beginning. Default: <code>false</code>
<b>margin</b> <b>marginleft</b> <b>marginbottom</b> <b>marginright</b> <b>margin<sup>1</sup>top</b>	(Float or percentage) Left/bottom/right/top cell margins in user coordinates (must be greater than or equal to 0) or as a percentage of the cell width or height (must be less than 100%). The specified margins define the inner cell box which serves as the fitbox for the cell contents. Default for <code>margin</code> : 0; Default for all others: <code>margin</code>
<b>minrow-height<sup>1</sup></b>	(Float or percentage) If a row cannot completely be placed in a table instance, this option specifies whether the row can be split and how small the fragments can get. The minimum fragment height can be specified in user coordinates or as a percentage of the row height. Default: 100%, i.e. no splitting
<b>repeatcontent</b>	(Boolean) Specify whether the contents of a table cell will be repeated if a cell or row is split between several table instances. Default: <code>true</code>  Splitting a cell: If the last rows spanned by a cell don't fit into the fitbox, the cell will be split. Except for <i>Textflows</i> (which will not be repeated), the cell contents will be repeated in the next table instance if <code>repeatcontent=true</code> . Otherwise it will not be repeated.  Splitting a row: If the last body row doesn't fit into the fitbox, it will usually not be split but will completely be placed in the next table instance. You can decrease the <code>minrowheight</code> value to split the last body row with the given percentage of contents in the first instance, and place the remaining parts of that row in the next instance. Except for <i>Textflows</i> (which will not be repeated), the cell contents will be repeated in the next table instance if <code>repeatcontent=true</code> . Otherwise it will not be repeated.

Table 5.15 Formatting options for `PDF_add_table_cell()`

key	explanation
<b>return</b> <sup>1</sup>	(String) <code>PDF_fit_table()</code> will stop after placing the specified row, and will return the specified string. The string must not start with an underscore character '_'. If the specified row is part of a join group it must be the last row of the group; otherwise an error will occur.
<b>rowheight</b> <sup>1</sup>	(Float or percentage) Height of the row specified in the <code>row</code> parameter. The height can be specified in user coordinates <sup>2</sup> , or as a percentage of the height of the table's first fitbox (see <code>PDF_fit_table()</code> ). User coordinates and percentages must not be mixed, i.e. either user coordinates or percentages must be used in all row height definitions of a table. The row height may be increased automatically if the row traverses cells containing text. Images, graphics and PDF pages in table cells don't have any influence on row heights. Default: see option <code>rowheightdefault</code> of <code>PDF_fit_table()</code>
<b>rowscale-group</b> <sup>1</sup>	(String) Name of a row group to which the row will be added. All rows in a group will be scaled uniformly if one of the rows in the group must be enlarged to completely hold long text. If a cell spans multiple rows the affected rows form a scale group automatically.
<b>rowjoin-group</b> <sup>1</sup>	(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

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.

Table 5.16 Options for static cell contents in `PDF_add_table_cell()` and suboptions for the caption option of `PDF_fit_table()`

key	explanation
<b>fitgraphics</b>	(Option list; only relevant for graphics) Option list for <code>PDF_fit_graphics()</code> . This option list will be applied to place the graphics supplied with the <code>graphics</code> option in the cell. The lower left corner of the fitbox will be used as reference point. Default: <code>fitmethod=meet position=center</code> . This option list is prepended to the user-specified options. <sup>1</sup>
<b>fitimage</b>	(Option list; only relevant for images and templates) Option list for <code>PDF_fit_image()</code> . This option list will be applied to place the image or template supplied with the <code>image</code> option in the cell. The lower left corner of the fitbox will be used as the reference point. Default: <code>fitmethod=meet position=center</code> . This option list is prepended to the user-specified options. <sup>1</sup>
<b>fitpath</b>	(Option list; only relevant for path objects) Option list for <code>PDF_draw_path()</code> . This option list will be applied to place the path object specified with the <code>path</code> option within its bounding box in the cell. The lower left corner of the fitbox will be used as reference point. Default: <code>fitmethod=meet position=center</code> . This option list is prepended to the user-specified options. <sup>1</sup>
<b>fitpdipage</b>	(Option list; only relevant for PDI pages; only if PDI is available) Option list for <code>PDF_fit_pdi_page()</code> . This option list will be applied to place the page supplied with the <code>pdipage</code> option in the cell. The lower left corner of the fitbox will be used as the reference point. Default: <code>fitmethod=meet position=center</code> . This option list is prepended to the user-specified options. <sup>1</sup>
<b>fittextflow</b>	(Option list; only relevant for Textflows) Option list for <code>PDF_fit_textflow()</code> . This option list will be applied to place the Textflow supplied in the <code>textflow</code> option in the cell. The fitbox will be used as fitbox. Default: <code>verticalalign=center lastlinedist=descender</code> . This option list will be prepended to the user-specified option list.

Table 5.16 Options for static cell contents in `PDF_add_table_cell()` and suboptions for the caption option of `PDF_fit_table()`

key	explanation
<b>fittextline</b>	(Option list; only relevant for Textlines) Option list for <code>PDF_fit_textline()</code> . This option list will be applied to fit the text supplied with the <code>text</code> parameter into the cell. The lower left corner of the fitbox will be used as the reference point. Options which have not been specified will be replaced with the respective defaults; the current text state is not taken into account. Default: <code>fitmethod=nofit position=center</code> . This option list is prepended to the user-specified options. <sup>1</sup>
<b>graphics</b>	(Graphics handle) The graphics associated with the handle will be placed in the fitbox.
<b>image</b>	(Image handle) The image or template associated with the handle will be placed in the fitbox.
<b>matchbox</b>	(Option list) Option list with matchbox details according to Table 6.4.
<b>path</b>	(Path handle) The path object within its bounding box will be placed in the fitbox according to the <code>fitpath</code> option.
<b>pdipage</b>	(Page handle) The imported PDF page associated with the handle will be placed in the fitbox.
<b>text</b>	(Content string) Text to be placed with <code>PDF_fit_textline()</code> according to the option <code>fittextline</code> . In <code>PDF_add_table_cell()</code> the value of this option can alternatively be provided via the function parameter <code>text</code> .
<b>textflow</b>	(Textflow handle) The Textflow associated with the handle will be placed in the fitbox. 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.

1. The box size is calculated automatically; any `boxsize` option in the supplied option list will be ignored.

Table 5.17 Options for interactive cell contents for `PDF_add_table_cell()` and suboptions for the caption option (only in page scope)

key	explanation
<b>annotation-type</b>	(String) Specifies the type of an annotation to be inserted in the table cell according to Table 12.2.
<b>fieldname</b>	(Hypertext string) Form field name for <code>fieldtype</code> .
<b>fieldtype</b>	(String) Specifies the type of a form field to be inserted in the table cell according to Table 12.4. Form field groups should be defined outside of tables.
<b>fitannotation</b>	(Option list) Annotation options for <code>annotationtype</code> according to Table 12.3.
<b>fitfield</b>	(Option list) Form field options for <code>fieldtype</code> according to Table 12.5.

---

C++ Java C# **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 Table 2.1)
- ▶ 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*
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*. This option can be used to trigger automatic table tagging (see PDFlib Tutorial for details).

**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 and set *debugshow=true* to visualize 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* 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 depending 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: areas which are defined by a *matchbox* definition.
- ▶ Contents: the specified cell contents will be placed in the following order: image, graphics, imported PDF page, graphics, path objects, Textflow, Textline, annotations, form fields.
- ▶ Matchbox ruling: 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, graphics etc. outside of the table functions.

Automatic table tagging: the *tag* option can be used to trigger automatic table tagging (see PDFlib Tutorial).

**Scope** Generally *page, pattern, template, glyph*; however, if the table contains form fields or annotations the respective scope of those table contents is dominant. For example, a table containing form fields or annotations cannot be placed on a template.

**PDF/UA** If automatic table tagging is active the table decoration (ruling and shading) is automatically tagged as Artifact.

Table 5.18 Options for `PDF_fit_table()`

key	explanation
<b>blind</b>	(Boolean) If true, all calculations will be performed, but no output will be created. The formatting results can be checked with <code>PDF_info_table()</code> . Default: false
<b>caption</b>	<p>(Option list) Create a fit box for a caption relative to the calculated fit box and fill it with various content types. The following option can be supplied (default: no caption):</p> <p><b>fitbox</b> (List of four floats or percentages with absolute or relative coordinates; required) Coordinates of two diagonal box corners in user coordinates. If a value is a percentage or a relative value it indicates the offset from the corresponding corner {llx lly urx ury} of the table instance. Percentages corresponding to llx or urx are percentages of the table instance width, percentages corresponding to lly or ury are percentages of the table instance height. The fitbox is not automatically adjusted to the size of its contents. The specified matchbox will describe the fitbox; this can be used to draw the caption fitbox or to retrieve the matchbox with <code>PDF_info_matchbox()</code>.</p> <p>Examples for using the fitbox option:  Fit box at the top of the table instance with a height of 20: <code>fitbox={0r 100% 0r 20r}</code>  Fit box to the right of the table instance with width 20 and offset 20% from the bottom:  <code>fitbox={100% 20% 20r 0r}</code></p> <p>In addition, the following options are supported:</p> <ul style="list-style-type: none"> <li>▶ Options for static cell contents according to Table 5.16: <code>fitgraphics</code>, <code>fitimage</code>, <code>fitpath</code>, <code>fitpdipage</code>, <code>fittextflow</code>, <code>fittextline</code>, <code>graphics</code>, <code>image</code>, <code>matchbox</code>, <code>path</code>, <code>pdipage</code>, <code>text</code>, <code>textflow</code></li> <li>▶ Options for interactive cell contents according to Table 5.17 (only in page scope): <code>annotationtype</code>, <code>fieldname</code>, <code>fieldtype</code>, <code>fitannotation</code>, <code>fitfield</code></li> <li>▶ Option for abbreviated structure element tagging according to Table 14.5: <code>tag</code>. This can be used for inserting a parent element of the caption contents, or a grouping element as container for multiple elements which comprise the caption contents.</li> </ul>
<b>colwidth-default</b>	<p>(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>):</p> <p><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.</p> <p><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>.</p> <p><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.</p> <p>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>).</p>
<b>debugshow</b>	(Boolean) If true, all errors for tables which are too high or too wide, or where the cells get too small are suppressed and logged instead. The resulting table instance is created as a debugging aid although the table is damaged. Default: false

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 <code>matchbox</code> option can be used to fill single cells with color, see Section 6.2, »Matchboxes«, page 131):</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 <code>col</code> in <code>PDF_add_table_cell()</code>)</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 <code>row</code> in <code>PDF_add_table_cell()</code>)</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 7.1 can also be used:  <b>fillcolor, shading</b></p> <p>Examples:            fill all rows in the table with red: <code>fill = { {area=table fillcolor=red} }</code>            fill odd-numbered rows with green and even-numbered rows with red:  <code>fill = { {area=rowodd fillcolor=green} {area=roweven fillcolor=red} }</code>            Use <code>fillcolor=none</code> 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: <code>vertlines</code>):</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 <code>PDF_create_gstate()</code>. All table decorations will be subject to the supplied graphics state. The cell contents will not be affected. Default: no <code>gstate</code> (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 <code>PDF_fit_table()</code>. Currently the following values are supported (default: 0):</p> <p><b>1</b> Rewind to the state before the first call to <code>PDF_fit_table()</code>.</p> <p><b>0</b> Don't reset the table.</p> <p><b>-1</b> Rewind to the state before the last call to <code>PDF_fit_table()</code> (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 <code>PDF_fit_table()</code> for a particular table) Default height of rows for which the <code>rowheight</code> option of <code>PDF_add_table_cell()</code> was not specified. The default height can be specified as an absolute value or as a keyword. If a float value is specified it is used as default row height unless it is smaller than the textbox height. The value 0 (zero) is equivalent to the keyword <code>distribute</code>. The following keywords are supported (default: <code>auto</code>):</p> <p><b>auto</b> Rows which contain only <code>Textline</code> cells have a height of two times the height of the textbox. The remaining height of the fitbox is distributed among all rows with <code>Textflow</code> or other cells. The table covers the full height of the fitbox.</p> <p><b>distribute</b> The height of the fitbox is 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 <code>Textline</code> cells have the height of the textbox, i.e. the smallest possible height to hold the text. Use the <code>boxsize</code> or <code>margin</code> options to increase the height of <code>Textline</code> 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 <code>Textline</code> or <code>Textflow</code> cells will be adjusted automatically (see <code>PDFlib Tutorial</code>).</p>
<b>showcells</b>	<p>(Boolean) If <code>true</code>, the border of each inner cell box will be stroked using the current graphics state. In page scope and if <code>PDF/A</code> is not active each cell is additionally decorated with an annotation with details describing the cell contents which may be helpful for analyzing table-related problems. Default: <code>false</code></p>
<b>showgrid</b>	<p>(Boolean) If <code>true</code>, the vertical and horizontal boundary of all columns and rows are stroked. Default: <code>false</code></p>
<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 #; <code>vert0</code> is the left table border</p> <p><b>vertfirst</b> first vertical line (equivalent to <code>vert0</code>)</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; <code>row0</code> 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 7.1 can also be used:  <b>dasharray, dashphase, linecap, linejoin, linewidth, strokecolor</b></p> <p>Examples:</p> <p>stroke all lines with black and <code>linewidth 1</code>: <code>stroke = {line=other}</code></p> <p>stroke the outer border lines with <code>linewidth 0.5</code>: <code>stroke = { {line=frame linewidth=0.5} }</code></p> <p>stroke the outer border lines with <code>linewidth 0.5</code>, and all other lines with <code>linewidth 0.1</code>:  <code>stroke = { {line=frame linewidth=0.5} {line=other linewidth=0.1} }</code></p> <p>Use <code>strokecolor=none</code> to suppress stroking for a table area.</p>
<b>vertshrink-limit</b>	<p>(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: <code>90%</code></p>

C++ Java C# **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:

- ▶ Keywords for querying the results of object fitting according to Table 6.3:  
*boundingbox, fitscalex, fitscaley, height, objectheight, objectwidth, width, x1, y1, x2, y2, x3, y3, x4, y4*
- ▶ Additional keywords according to Table 5.19:  
*firstbodyrow, horboxgap, horshrinking, lastbodyrow, returnreason, rowcount, rowsplit, tableheight, tablewidth, vertboxgap, vertshrinking, xvertline#, yhorline#*,

**Returns** The value of some table parameter as requested by *keyword*. This function returns correct geometry information even in blind mode. If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with `PDF_get_string()`.

**Scope** any except *object*

Table 5.19 Keywords for `PDF_info_table()`

<b>keyword</b>	<b>explanation</b>
<b>firstbodyrow</b>	Number of the first body row in the most recently placed 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 for 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>tableheight</b> <b>tablewidth</b>	Width and height of the entire table
<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>xvertline#</b>	x coordinate of the vertical line with number #. <i>xvertline0</i> is the left table border.
<b>yhorline#</b>	y coordinate of the horizontal line with number #. <i>yhorline0</i> is the top table border.



---

C++ Java C# **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 Option for `PDF_delete_table()`

key	explanation
<b>keephandles</b>	(Boolean) If false, all handles supplied to the <code>textflow</code> , <code>image</code> , <code>graphics</code> and <code>pdipage</code> options of <code>PDF_add_table_cell()</code> 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_graphics()`, `PDF_info_graphics()`
- ▶ `PDF_fit_pdi_page()`, `PDF_info_pdi_page()`
- ▶ `PDF_draw_path()`, `PDF_info_path()`
- ▶ `PDF_add_table_cell()` (via option lists for the *fitgraphics*, *fitimage*, *fitpdipage*, *fitpath*, *fittextline* 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 5.2, »Multi-Line Text with *Textflows*«, page 95.

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 width is the width of the text string (in horizontal writing mode) or the width of the widest glyph (in vertical writing mode). 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. Character spacing will not be applied after the last glyph.
- ▶ 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.
- ▶ Graphics (`PDF_fit/info_graphics()`): the suboption *clipping* of the *matchbox* option can be used to define some part of the object as object box. The object box is defined by the width and height of the SVG graphics or by *forcedwidth* and *forcedheight*. If these values are 0 the following holds: if *fitmethod* is different from *nofit* or the *fitbox* is not defined, the size of the object box is defined by *fallbackwidth* and *fallbackheight*. If *fitmethod=nofit* and the *fitbox* is defined, the size of the object box is defined by the *fitbox*.

- ▶ Imported PDF pages (*PDF\_fit/info\_pdi\_page()*, PDF Blocks, table cells): the options used in *PDF\_open\_pdi\_page()* will be honored. If *cloneboxes=true* the visible box will be used (i.e. the CropBox if present, else the MediaBox). 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. The *linewidth* and *miterlimit* options will be ignored.
- ▶ 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/urx/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, graphics, 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={0 0}* 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 algorithm according to Table 6.2.

In this case the options *scale* and *dpi* are ignored, and the options *margin*, *shrinklimit*, and *showborder* are 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. The path object will be placed so that the specified attachment point coincides with the reference point. If fitmethod is different from nofit the object will first be placed in the fitbox according to the specified method. 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

Table 6.1 Fitting options for various functions

option	explanation
<b>boxsize</b>	<p>(List of two 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 is 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.</p> <p>Default for Blocks: width and height of the Block's Rect property            Default for all other fitting functions: {0 0}</p>
<b>dpi</b>	<p>(List of two floats or keywords; 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 is used for both dimensions. With the value zero the image's internal resolution is 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 is different from the supplied values. The scale option will be applied in addition to the dpi values.</p> <p>If the fitmethod option has been supplied with one of the keywords auto, meet, slice, or entire, the dpi values specify only the image's aspect ratio, but not its absolute size. Default: internal</p>
<b>fitmethod</b>	<p>(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.</p> <p>Default: clip for Textflow; meet for tables, path objects and reference option; and nofit otherwise</p>
<b>margin</b>	<p>(List of floats; only for Textlines) One or two float values describing additional horizontal and vertical reductions of the fitbox. Default: 0</p>
<b>matchbox</b>	<p>(Option list; not for path objects) Option list for creating a matchbox according to Table 6.4</p>
<b>minfontsize</b>	<p>(Float or percentage; only for Textflow) Minimum allowed font size when text is scaled down to fit into the fitbox with fitmethod=auto when shrinklimit 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 minfontsize as fontsize. Default: 0.1%</p>
<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)  <b>east</b>    pointing to the right (270)  <b>south</b>   upside down (180)  <b>west</b>    pointing to the left (90)</p>

Table 6.1 Fitting options for various functions

option	explanation
<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.</p> <p>Default: {0 100} for tables, center for the reference option, otherwise {0 0}. 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 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. Examples:</p> <p>{left center} or {0 50} left-aligned          {center} or {50 50} centered          {right center} or {100 50} right-aligned</p> <p>Only for Textlines: the keyword auto can be used for the first value in the list. It indicates right if the writing direction of the text is from right to left (e.g. for Arabic and Hebrew text), and left otherwise</p>
<b>repoint</b>	<p>(List of floats; only for PDF_fill_*block() and PDF_info_path()) Specifies the reference point in user coordinates for fitting the block contents or path.</p> <p>Default for PDF_fill_*block(): lower left corner of the rectangle defined by the Block's Rect property</p> <p>Default for PDF_info_path(): {0 0}</p>
<b>rotate</b>	<p>(Float; not for tables and 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</p> <p>Textline in table cells: if the rotate option was specified with a value different from 0, the table engine attempts to fit the bounding box of the rotated text into the cell box according to the fitmethod and position options. If fitmethod is different from auto, the cell will be enlarged appropriately if necessary.</p>
<b>scale</b>	<p>(List of floats; not for Textlines and Textflow) 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. Negative values result in mirroring. The absolute value of this option ignored if the fitmethod option has been supplied with one of the keywords auto, meet, slice, or entire. Default: {1 1}</p>
<b>stamp</b>	<p>(Keyword; only for Textlines; will be ignored if boxsize is not specified) This option can be used to create a diagonal stamp of maximal size in the rectangle specified with the boxsize 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 fontsize, fitmethod, and position will be ignored. The options orientate=west and =east don't make any sense (only north and south). Supported keywords (default: none):</p> <p><b>ll2ur</b> The stamp runs diagonally from the lower left corner to the upper right corner.</p> <p><b>ul2lr</b> The stamp runs diagonally from the upper left corner to the lower right corner.</p> <p><b>none</b> No stamp will be created.</p>

Table 6.1 Fitting options for various functions

<b>option</b>	<b>explanation</b>
<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 <code>Textlines</code> ) 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>This method tries to fit the object box into the fitbox automatically: If the object fits into the fitbox the behavior is identical to the nofit method, i.e. the object is placed without any scaling.</p> <p>If the object is larger than the fitbox the object is proportionally reduced in size as follows:</p> <p>Textlines: a scaling factor is calculated such that the text can 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> <p>Other object types: the behavior is identical to the meet method.</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 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 option will be applied to images and graphics, for images also the dpi option.</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>	

**Common keywords for querying the results of object fitting.** The results of object fitting can be queried without actually placing the object on the page. This can be used to make formatting decisions before actually creating page content. In order to query formatting results the fitting options for an object can be supplied to the respective `PDF_info_*`( ) function. Table 6.3 lists keywords for querying fitting results. The fitting results for `PDF_info_path()` are expressed relative to the reference point.

Table 6.3 Common keywords for querying the results of object fitting with `PDF_info_image()`, `PDF_info_graphics()`, `PDF_info_path()`, `PDF_info_pdi_page()`, `PDF_info_table()`, `PDF_info_textline()`

<b>keyword</b>	<b>explanation</b>
<b>boundingbox</b>	Path handle for the object's bounding box
<b>fitscalex, fitscaley</b>	Scaling factors which resulted from fitting the object to a box.
<b>height</b>	Object height in user coordinates
<b>objectheight, objectwidth</b>	Raw size of the object after processing all options relevant for loading or creating the object. This size will be used by the fitting algorithm.
<b>width</b>	Object width in user coordinates
<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 object's bounding box in user coordinates according to the supplied options.

## 6.2 Matchboxes

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

- ▶ Textlines with *PDF\_fit\_textline()*, *PDF\_fill\_textblock()* with *textflow=false*: the matchbox describes the bounding box of the text line.
- ▶ Textflows with *PDF\_add/create\_textflow()*, *PDF\_fit\_textflow()*, *PDF\_fill\_textblock()* with *textflow=true*: the matchbox describes the bounding box of the generated text output. Matchbox specifications in *PDF\_fill\_textblock()* cannot be used as start for inline text decorations, but only for creating a matchbox for the whole text.
- ▶ imported PDF pages with *PDF\_fit\_pdi\_page()*, *PDF\_fill\_pdf\_block()*: the matchbox describes the bounding box of the placed page.
- ▶ images and templates with *PDF\_fit\_image()*, *PDF\_fill\_image\_block()*: the matchbox describes the bounding box of the placed image or template.
- ▶ graphics with *PDF\_fit\_graphics()*: the matchbox describes the bounding box of the placed graphics.
- ▶ table cells: *PDF\_add\_table\_cell()*: the matchbox describes the bounding box of the 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 7.1:  
*borderwidth, dasharray, dashphase, fillcolor, gstate, linecap, linejoin, shading, strokecolor*
- ▶ Matchbox controlling options according to Table 6.4;
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

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

Table 6.4 Suboptions for the matchbox option of various functions

option	explanation
<b>boxheight</b>	(List with two elements, each being a positive float, a percentage of the fontsize, or a keyword; only for <i>Textline</i> and <i>Textflow</i> ) 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), <i>xheight</i> , <i>descender</i> , <i>capheight</i> , <i>ascender</i> , <i>fontsize</i> , <i>leading</i> , <i>textrise</i> With <i>Textflows</i> the values corresponding to the text at the beginning of the matchbox will be used. Default: { <i>capheight</i> none}
<b>boxwidth</b>	(Float or percentage; only for <i>Textflow</i> ) Width of the matchbox specified in user coordinates or as a percentage of the width of the fitbox. If this option is supplied, horizontal space of the specified width is 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 <i>Textflow</i> . Note that with <i>alignment=justify</i> the box width may be compressed the same way as text (see option <i>shrinklimit</i> ). Default: 0

Table 6.4 Suboptions for the matchbox option of various functions

option	explanation
<b>clipping</b>	<p>(Rectangle or 4 percentages; only for images, graphics and imported PDF pages; will be ignored if the <code>innerbox</code> option has been specified) Coordinates of the lower left and upper right corner of a rectangle within the image, graphics or page specifying which part should be displayed. The specification depends on the type of object (default: {0% 0% 100% 100%}):</p> <ul style="list-style-type: none"> <li>▶ For images the clipping rectangle can be specified in pixels or as a percentage of the width/height.</li> <li>▶ For graphics the clipping rectangle can be specified in user coordinates or as a percentage of the width/height of the graphics' object box.</li> <li>▶ For 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.</li> </ul>
<b>create-wrapbox</b>	<p>(Boolean; only for Textflow) If <code>true</code>, 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: <code>false</code></p>
<b>doubleadapt</b>	<p>If <code>true</code> 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 <code>doubleoffset</code>. Default: <code>true</code></p>
<b>doubleoffset</b>	<p>(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)</p>
<b>drawleft</b> <b>drawbottom</b> <b>drawright</b> <b>drawtop</b>	<p>(Boolean) If <code>true</code>, the corresponding border of the rectangle will be drawn provided that the <code>borderwidth</code> is set to a value greater than 0. Default: <code>true</code></p>
<b>end</b>	<p>(Boolean; only for Textflow) Specifies the end of the matchbox. If <code>true</code>, 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 <code>boxwidth</code> (if specified) and the extent of the text enclosed in the options <code>matchbox</code> and <code>matchbox= end</code>. If the <code>end</code> option has not been specified, the matchbox will end after the last character in the Textflow.</p>
<b>exceedlimit</b>	<p>(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 <code>PDF_fit_textflow()</code> will return <code>_boxfull</code>; 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.</p>
<b>innerbox</b>	<p>(Boolean; only for table cells, and TIFF and JPEG images) Table cells: If <code>true</code>, 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 <code>true</code> and the image contains a clipping path the bounding box of the clipping path will be used instead of the full image. Default: <code>false</code></p>
<b>margin</b>	<p>(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 <code>offset*</code> has been supplied. Default: 0</p>
<b>name</b>	<p>(Name string) Name of the matchbox. If the name has already been assigned to a matchbox, an additional rectangle for this matchbox will be created. This means that a matchbox may consist of more than one rectangle. The name can be used in <code>PDF_info_matchbox()</code>. Various functions support the option <code>usematchbox</code> to reference one or more rectangles of a matchbox, e.g. to add an annotation with <code>PDF_create_annotation()</code>. Matchbox names can be used until the end of the current page. The name <code>»*«</code> (asterisk character) should not be used as matchbox name. Default: <code>no name</code></p>

Table 6.4 Suboptions for the matchbox option of various functions

option	explanation
<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 has to be split (e.g. because of a font change or line break), 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 C# **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 a matchbox which has been created under this name on the current page. It must have been created with the *name* suboption of the *matchbox* option when the matchbox was defined. Alternatively, the name *''* (asterisk character) can be used to query information about all matchboxes on the page. An empty *boxname* can be used to query information about all matchbox rectangles on the current page.

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

**num** Positive number of a matchbox or rectangle (the first has number 1).

**keyword** A keyword specifying the requested information according to Table 6.5.

**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, the return value is -1 (in PHP: 0) for the keywords *boundingbox*, *name*, and *rectangle*, and 0 for all other keywords. If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with *PDF\_get\_string()*.

Depending on the current scope, the function returns information about the matchboxes on the current page, pattern, template, or glyph description.

**Details** Named matchboxes within a Textflow can only be queried after calling *PDF\_fit\_textflow()*. Matchboxes created in blind mode cannot be queried.

Rectangles for the keywords *boundingbox*, *exists*, *height*, *name*, *rectangle*, *width*, *x1*, *y1*, ..., *x4*, *y4* are selected as follows:

- ▶ If *boxname* contains the name of a matchbox: select the *num*-th rectangle of the specified named matchbox on the current page.

- ▶ If *boxname*=\*: select the first rectangle of the *num*-th named matchbox on the current page.
- ▶ If *boxname* is empty: select the *num*-th rectangle created by a named matchbox on the current page.

*Scope* any except *document* and *object*

Table 6.5 Keywords for `PDF_info_matchbox()`

<b>keyword</b>	<b>explanation</b>
<b>boundingbox</b>	Handle of a path object containing the bounding box of the selected rectangle in the current user coordinate system or -1 (in PHP: 0) if the specified rectangle doesn't exist. The bounding box is different from the rectangle if the matchbox was rotated.
<b>count</b>	(The <i>num</i> parameter will be ignored) If <i>boxname</i> contains the name of a matchbox: Number of rectangles for this matchbox If <i>boxname</i> =*: number of matchboxes with at least one rectangle If <i>boxname</i> is empty: total number of rectangles created by named matchboxes
<b>exists</b>	1 if the selected rectangle exists, 0 otherwise.
<b>height<sup>1</sup></b>	Height of the selected rectangle in user coordinates
<b>name</b>	String index for the name of the matchbox for which the selected rectangle was created. The corresponding string can be retrieved via <code>PDF_get_string()</code>
<b>rectangle</b>	Handle of the path object containing the selected rectangle in user coordinates or -1 (in PHP: 0) if the rectangle couldn't be found
<b>width<sup>1</sup></b>	Width of the selected rectangle in user coordinates
<b>x1, y1, ... , x4, y4<sup>1</sup></b>	Position of the <i>i</i> -th corner ( <i>i</i> =1, 2, 3, 4) of the selected rectangle in user coordinates. In the coordinate system of the respective fit element (image, text, etc.), x1, y1 correspond to the lower left, x2, y2 to the lower right, x3, y3 to the upper right and x4, y4 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 Options

**Graphics appearance options.** The graphics appearance options in Table 7.1 can be used with the following functions (note that not all functions support all options; see function descriptions for details):

- ▶ `PDF_set_graphics_option()`
- ▶ `PDF_create_gstate()` (only `flatness`, `linecap`, `linejoin`, `linewidth`, `miterlimit`)
- ▶ `PDF_add_path_point()` and `PDF_draw_path()`
- ▶ The `fill` option of `PDF_fit_table()` (only `fillcolor`, `shading`) and the `stroke` option of `PDF_fit_table()` (only `dasharray`, `dashphase`, `linecap`, `linejoin`, `linewidth`, `strokecolor`)
- ▶ The `matchbox` option of various functions

Table 7.1 Graphics appearance options

option	explanation and possible values
<code>cliprule</code>	(Keyword) Clipping rule which determines the interior of areas for clipping; see <code>fillrule</code> for possible keywords. Default: value of the <code>fillrule</code> option
<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 two non-negative floats or keyword) List of 2-12 alternating values for the lengths of dashes and gaps for stroked paths (measured in the user coordinate system). The array values must not be negative. They will be cyclically reused until the complete path is stroked. The keyword <code>none</code> can be used to create solid lines. Default: none
<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: <code>winding</code> ): <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. <b>evenodd</b> Use the even-odd rule, which yields the same results as <code>winding</code> for simple shapes, but produces different results for more complex shapes, especially self-intersecting paths.
<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

Table 7.1 Graphics appearance options



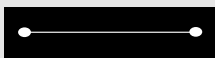


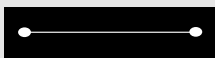


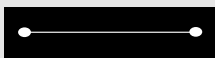









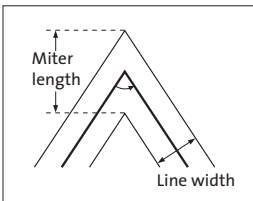
option	explanation and possible values									
<b>gstate</b>	(Gstate handle) Handle for a graphics state retrieved with <code>PDF_create_gstate()</code> . Default: no graphics state (i.e. current settings will be used)									
<b>initgraphics-state</b>	(Boolean; only for <code>PDF_set_graphics_option()</code> ) If true all graphics appearance options are initialized with the default values. The current clipping path is not affected. If false the current graphics state values are used. Default: false									
<b>linecap</b>	(Integer or keyword) Shape at the end of a path (default: projecting in <code>PDF_fit_table()</code> , otherwise butt): <table border="0" style="width: 100%; margin-top: 10px;"> <tr> <td style="width: 15%;"><b>butt</b></td> <td style="width: 65%;">(Equivalent value: 0) Butt end caps: the stroke is squared off at the endpoint of the path.</td> <td style="width: 20%;"></td> </tr> <tr> <td><b>round</b></td> <td>(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.</td> <td></td> </tr> <tr> <td><b>projecting</b></td> <td>(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.</td> <td></td> </tr> </table>	<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>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): <table border="0" style="width: 100%; margin-top: 10px;"> <tr> <td style="width: 15%;"><b>miter</b></td> <td style="width: 65%;">(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.</td> <td style="width: 20%;"></td> </tr> <tr> <td><b>round</b></td> <td>(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.</td> <td></td> </tr> <tr> <td><b>bevel</b></td> <td>(Equivalent value: 2) Bevel joins: the two path segments are drawn with butt end caps (see the discussion of linecap), and the resulting notch beyond the ends of the segments is filled in with a triangle.</td> <td></td> </tr> </table>	<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 linecap), and the resulting notch beyond the ends of the segments is filled in with a triangle.	
<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 linecap), 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. <div style="text-align: right; margin-top: 10px;">  </div>									
<b>shading</b>	(Option list according to Table 7.2; 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 7.2 Suboptions for the shading graphics appearance option

option	explanation
<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 variable <i>t</i> . 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 <i>x</i> and <i>y</i> 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 <i>x</i> and <i>y</i> 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

C++ Java C# **void set\_graphics\_option(String optlist)**

Perl PHP **set\_graphics\_option(string optlist)**

C **void PDF\_set\_graphics\_option(PDF \*p, const char \*optlist)**

Set one or more graphics appearance options.

**optlist** An option list specifying graphics appearance options according to Table 7.1.

The following options can be used:

*cliprule, dasharray, dashphase, fillcolor, fillrule, flatness, gstate, initgraphicsstate, linecap, linejoin, linewidth, miterlimit, strokecolor*

**Details** Graphics appearance options set the graphics state for the following groups of functions:

- ▶ explicit drawing functions, e.g. *PDF\_stroke()*, *PDF\_fill()*
- ▶ implicit drawing functions, e.g. the *showborder* option of *PDF\_fit\_textline()*, *PDF\_fit\_textflow()*
- ▶ text output created with simple text output functions if no color has been set with text options, e.g. *PDF\_show()*

All graphics appearance options are reset to their default values at the beginning of a page, pattern, template, or glyph description, and retain their values until the end of the current *page*, *pattern*, *template*, or *glyph* scope. However, the graphics appearance options can also be reset with the *initgraphicsstate* option.

A subsequent call to *PDF\_setcolor()* overrides the *fillcolor* and/or *strokecolor* values. A subsequent call to *PDF\_setlinewidth()* overrides the *linewidth* value.

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

## 7.2 Graphics State

---

C++ Java C# **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** This function sets the line width in the graphics state (see [PDF\\_set\\_graphics\\_option\(\)](#)) as well as the stroke width in the text state (see [PDF\\_set\\_text\\_option\(\)](#)). The *width* is reset to the default value of 1 at the beginning of each page.

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

---

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

Perl PHP **save()**

C **void PDF\_save(PDF \*p)**

---

Save the current graphics state to a stack.

**Details** The graphics state contains options 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 options explicitly again. The following items are subject to save/restore:

- ▶ graphics appearance options:  
clipping path, coordinate system, current point, flatness tolerance, line cap style, dash pattern, line join style, line width, miter limit;
- ▶ color options: fill and stroke colors;
- ▶ graphics options which have been set with explicit graphics states in [PDF\\_set\\_gstate\(\)](#);
- ▶ text position and the following text appearance options:  
*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.

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

**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.

---

---

C++ Java C# **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 C# **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 7.1:  
*flatness, linecap, linejoin, linewidth, miterlimit*
- ▶ Graphics state options according to Table 7.3:  
*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 options.

**Scope** any except *object*

---

C++ Java C# **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*

---

Table 7.3 Options for `PDF_create_gstate()`

key	explanation and possible values
<b>alphaisshape</b>	(Boolean) Sources of alpha are treated as shape (true) or opacity (false). Default: false
<b>blendmode</b>	(Keyword list; in PDF/X-1/3 and 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; in PDF/A-1 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; in PDF/A-1 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 fill operations. Default: false
<b>overprintmode</b>	(Integer) Overprint mode which modifies the overprint behavior of elements in DeviceCMYK color if a color component is 0. The overprint mode affects text and vector elements which are painted in DeviceCMYK color, but not images or shading patterns. Overprint mode 0 (zero) means that each color component replaces previously placed marks («foreground color wins»). Overprint mode 1 means that a color component of 0 leaves the corresponding component unchanged («foreground tint value 0 is ignored»). PDF/A-2/3: overprintmode=1 is not allowed if the current color space is ICC-based CMYK and overprintfill or overprintstroke is true. Default: 0
<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. Supported options and keyword (default: none): <b>backdropcolor</b> (List with one, three, or four floats; only 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>none</b> (Keyword) No soft mask at all; this is required to disable soft masks which may be in effect from a previously set graphics state. <b>template</b> (Template handle; required) Transparency group template which has been created with <code>PDF_begin_template_ext()</code> and the transparencygroup option. If type=luminosity the template must have been created with the colorspace suboption and a value different from none. <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) 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 C# **void setdash(double b, double w)**

Perl PHP **setdash(float b, float w)**

C **void PDF\_setdash(PDF \*p, double b, double w)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

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

Perl PHP **setdashpattern(string optlist)**

C **void PDF\_setdashpattern(PDF \*p, const char \*optlist)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

C++ Java C# **void setflat(double flatness)**

Perl PHP **setflat(float flatness)**

C **void PDF\_setflat(PDF \*p, double flatness)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

C++ Java C# **void setlinejoin(int linejoin)**

Perl PHP **setlinejoin(int linejoin)**

C **void PDF\_setlinejoin(PDF \*p, int linejoin)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

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

Perl PHP **setlinecap(int linecap)**

C **void PDF\_setlinecap(PDF \*p, int linecap)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

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

Perl PHP **setmiterlimit(float miter)**

C **void PDF\_setmiterlimit(PDF \*p, double miter)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

C++ Java C# **void initgraphics()**

Perl PHP **initgraphics()**

C **void PDF\_initgraphics(PDF \*p)**

---

Deprecated, use [PDF\\_set\\_graphics\\_option\(\)](#).

---

## 7.3 Coordinate System Transformations

All transformation functions (*PDF\_translate()*, *PDF\_scale()*, *PDF\_rotate()*, *PDF\_align()*, *PDF\_skew()*, *PDF\_concat()*, *PDF\_setmatrix()*, and the *initgraphicsstate* option of *PDF\_set\_graphics\_option()*) change the coordinate system used for drawing subsequent objects. They do not affect any existing objects on the page.

---

C++ Java C# **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 C# **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

*Bindings* COM: this function is also available under the name *pscale* to work around a bug in VB.

---

C++ Java C# **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 C# **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 C# **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 C# **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 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 C# **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 replaces it with the new matrix.

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



## 7.4 Path Construction

*Note* Make sure to call one of the functions in Section 7.5, »Painting and Clipping«, page 149, after using the functions in this section, or the constructed path will have no effect, and subsequent operations may raise an exception.

PDF/UA Vector graphics must be tagged as *Artifact* or *Figure* with a call to `PDF_begin_item()`.

---

C++ Java C# **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, glyph, path*; this function starts *path* scope.

---

C++ Java C# **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* option) will be painted on each side of the line connecting both endpoints. The behavior at the endpoints is determined by the *linecap* option.

*Scope* *path*

---

C++ Java C# **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  $(x3, y3)$ , using  $(x1, y1)$  and  $(x2, y2)$  as control points. The current point must be set before using this function. The endpoint of the curve becomes the new current point.

*Scope* *path*

---

---

C++ Java C# **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. The circle is created in counterclockwise direction.

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

**Bindings** COM: this function is also available under the name *pcircle* to work around a bug in VB 6.

---

C++ Java C# **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, glyph, path*; this function starts *path* scope.

---

C++ Java C# **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 C# **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*

---

C++ Java C# **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. The ellipse is created in counterclockwise direction.

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

---

C++ Java C# **void elliptical\_arc(double x, double y, double rx, double ry, String optlist)**

Perl PHP **elliptical\_arc(float x, float y, double rx, double ry, string optlist)**

C **void PDF\_elliptical\_arc(PDF \*p, double x, double y, double rx, double ry, const char \*optlist)**

---

Draw an elliptical arc segment from the current point.

**x, y** The coordinates of the end point of the elliptical arc segment.

**rx, ry** The  $x$  and  $y$  radii of the ellipse. At least one of these values must be larger than half the distance between the current point and  $(x, y)$ .

**optlist** An option list specifying construction options for the elliptical arc according to Table 7.4.

**Details** This function adds an elliptical arc segment to the current path. The arc segment will start at the current point and end at  $(x, y)$ . The current point must be set before using this function. The end point of the arc becomes the new current point. Two of the four possible arc segments represent an arc segment of  $\leq 180^\circ$  (the small arc segments), while the other two represent an arc segment of  $\geq 180^\circ$  (the large arc segments).

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

Table 7.4 Options for `PDF_elliptical_arc()`

<b>option</b>	<b>explanation</b>
<b>clockwise</b>	(Boolean) If <code>true</code> one of the clockwise arc segments will be created; otherwise one of the counterclockwise arc segments will be created. Default: <code>false</code>
<b>largearc</b>	(Boolean) If <code>true</code> one of the large arc segments will be created; otherwise one of the small arc segments will be created. Default: <code>false</code>
<b>rectify</b>	(Boolean) If <code>true</code> radii which are too small will be modified so that the elliptical arc can be constructed; otherwise an exception will be thrown. Default: <code>false</code>
<b>xrotate</b>	(Float) Rotation angle for the ellipse, i.e. the angle of the ellipse x axis relative to the current coordinate system x axis in degrees. The start and end point of the arc segment remain fixed. Default: <code>0</code>

---

C++ Java C# **`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 linewidth (as determined by the value of the *linewidth* option) will be painted on each side of the line connecting the respective endpoints. The rectangle is created in counterclockwise direction.

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

---

C++ Java C# **`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*

## 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 C# **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 C# **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 C# **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* option). Open paths are implicitly closed before being filled.

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

---

C++ Java C# **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.

---

---

C++ Java C# **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.

---

C++ Java C# **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\(\)](#). The clipping area is determined according to the algorithm selected with the *cliprule* option.

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

---

C++ Java C# **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 C# `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 or path 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: `o`) 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=``circle`, `circular`, `elliptical`, `ellipse`, `move`, `line`, `curve`, `rect`.

**type** Specifies the type of the point according to Table 7.5.

Table 7.5 Types of points for `PDF_add_path_point()`

keyword	explanation
<b>addpath</b>	Add the path definition specified in the <code>svgpath</code> option as a complete subpath, using ( $x, y$ ) as origin.
<b>circle</b>	Add a circle to the path as a complete subpath, using ( $x, y$ ) as the center and <code>radius</code> for the size. <sup>1</sup>
<b>circular</b>	Add a circular arc from the current point to ( $x, y$ ) 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. <sup>2</sup>
<b>control</b>	Control point for a Bézier curve or a circular arc. <sup>2</sup>
<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. <sup>2</sup>
<b>ellipse</b>	Add an ellipse to the path as a complete subpath, using ( $x, y$ ) as the center and the values in the <code>radius</code> option for the size. <sup>1</sup> The ellipse can be rotated with the <code>xrotate</code> option.
<b>elliptical</b>	Add an elliptical arc from the current point to ( $x, y$ ). The size and orientation of the ellipse are defined by the <code>radius</code> , <code>xrotate</code> , <code>largearc</code> , and <code>clockwise</code> options. If only a single value is provided as <code>radius</code> a circular arc will be created. In this case an appropriate circular arc point will be created automatically. If two values are provided in the <code>radius</code> option a set of Bézier curves will be created. <sup>2</sup>
<b>line</b>	Add a line segment from the current point to ( $x, y$ ). <sup>2</sup>
<b>move</b>	Start a new subpath. Subpaths will be numbered consecutively (1, 2, ...). The first subpath starts at the origin.
<b>pathref</b>	Add a reference to the path specified in the <code>path</code> option as a complete subpath, using ( $x, y$ ) as origin. Since the path is referenced (and not copied) subsequent changes to path will be reflected when drawing the path.
<b>rect</b>	Add a rectangle to the path as a complete subpath, using ( $x, y$ ) as the center of the rectangle and <code>width</code> and <code>height</code> for the size. <sup>1</sup> The corners of the rectangle can be rounded before drawing with the <code>round</code> options. Alternatively, the corners can be rounded with elliptical arcs with the <code>radius</code> option.

1. A new point with `type=move` and the same coordinates and graphics appearance options is created automatically after the path.

2. Graphics appearance options and path operation options are not allowed for these types.

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

- ▶ Path calculation and naming options for a point according to Table 7.6:  
*name, polar, radians, relative*
- ▶ Path operation options according to Table 7.6:  
*close, fill, round, stroke*
- ▶ Options for adding path definitions according to Table 7.6:  
*path, svgpath*
- ▶ Options for constructing path elements according to Table 7.6:  
*clockwise, height, largearc, radius, rectify, width, xrotate*
- ▶ Graphics appearance options according to Table 7.1 (only for *type=addpath, circle, ellipse, move, rect, or pathref*):  
*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, where new path elements can be created by specifying individual path nodes or by adding path definitions specified via a path handle or an SVG path description. The generated path can later be used with `PDF_draw_path()` and other functions.

A path object can hold any number of paths. Each path in turn may contain one or more subpaths which can be selected for drawing in the *subpaths* option of `PDF_draw_path()`. All paths will be closed, filled, stroked, and rounded separately according to the specified options.

An operation with any of the types *addpath, circle, ellipse, move, rect, or pathref* starts a new subpath. Graphics appearance options and path operation options (e.g. *stroke, fill*) can only be changed for *type=addpath, circle, ellipse, move, rect, or pathref*. In this situation a new path within the path object will be started automatically. Shapes of type *circle, ellipse, elliptical, and rect* are created in counterclockwise direction by default, but this can be changed with the option *clockwise*.

**Scope** any

Table 7.6 Options for `PDF_add_path_point()`

option	explanation
<b>clockwise</b>	(Boolean; only for <i>type=circle, ellipse, elliptical, rect</i> ) If true the shape is created in clockwise direction, otherwise counterclockwise. Default: false
<b>close</b>	(Boolean; only for <i>type=move</i> ) If true, the subpath will be closed with a straight line. Default: see footnote <sup>1</sup>
<b>fill</b>	(Boolean; only for <i>type=move</i> ) If true the subpath will be closed and filled. Default: see footnote <sup>1</sup>
<b>height</b>	(Float; only for <i>type=rect</i> ; required in this case) Height of the rectangle
<b>largearc</b>	(Boolean; only for <i>type=elliptical</i> ) If true one of the large elliptical arc segments will be created; otherwise one of the small elliptical arc segments will be created. Default: false
<b>name</b>	(String) Name of the point. Default: <code>p&lt;i&gt;</code> (e.g. <code>p1</code> ) where <i>i</i> is the consecutive number of supplied points.
<b>path</b>	(Path handle; only for <i>type=pathref</i> ) The specified path will be added to the current path by reference. The coordinates of the added path refer to the current point as origin. Graphics appearance options and the name option will be ignored.



Table 7.6 Options for `PDF_add_path_point()`

option	explanation
<b>polar</b>	(Boolean) If true, the (x, y) parameters are polar coordinates specifying radius r and angle phi, otherwise Cartesian coordinates specifying x and y values. Default: false
<b>radians</b>	(Boolean) If true, angles for polar coordinates are specified in radians, otherwise in degrees. Default: false
<b>radius</b>	(One or two floats; required for type=circle, ellipse and elliptical; also allowed for type=rect) The first value specifies the radius of the circle or the x radius of the ellipse. The second float value, if present, specifies the y radius of the ellipse. The first value will be used as default for the second value. For type=rect the values specify the x and y radii of the elliptical arcs in the rectangle corners. The elliptical arcs will be created immediately. Default: 0
<b>rectify</b>	(Boolean; only for type=ellipse and elliptical) If true radii which are too small will be modified so that the elliptical arc can be constructed; otherwise an exception will be thrown. Default: false
<b>relative</b>	(Boolean) If true, (x, y) are relative to the current point, otherwise to the current origin. Default: see footnote <sup>1</sup>
<b>round</b>	(Float; only for type=move) 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 close=true 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 round=0 no rounding will be done. The circular arcs will be created when the path is drawn. Default: see footnote <sup>1</sup>
<b>stroke</b>	(Boolean; only for type=move) If true the subpath will be stroked. Default: see footnote <sup>1</sup>
<b>svgpath</b>	(String; only for type=addpath) String containing a path description in SVG syntax according to www.w3.org/TR/SVG11/paths.html#PathData. The specified path will be added to the current path. The coordinates of the specified SVG path refer to the current point as origin. Graphics appearance options can be specified for the SVG path. The option rectify is taken into account for the inserted SVG path. If the path stems from an SVG file with a topdown coordinate system it must be mirrored (even if PDFlib operates in topdown mode). This can be achieved with the option scale={1 -1} in <code>PDF_draw_path()</code>
<b>width</b>	(Float; only for type=rect; required in this case) Width of the rectangle
<b>xrotate</b>	(Float; only for type=ellipse and elliptical) Rotation angle for the ellipse in current units (see option radians), i.e. the angle of the ellipse x axis relative to the current coordinate system x axis in degrees. The start and end point of the arc segment remain fixed. This option will be ignored if only a single value was supplied as radius. Default: 0

1. 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 C# **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 7.7:  
*clip, close, fill, round, stroke, subpaths*
- ▶ Options for modifying the bounding box according to Table 7.7:  
*bboxexpand, boundingbox*
- ▶ Graphics appearance options for the *fill* and *stroke* options according to Table 7.1:  
*dasharray, dashphase, fillcolor, flatness, gstate, linecap, linejoin, linewidth, miterlimit, strokecolor*
- ▶ Graphics appearance options according to Table 7.1 for the *clip* option according to Table 7.1: *cliprule, fillrule*
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

**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()`.

**PDF/UA** All path objects must be tagged as *Artifact* or *Figure*, either with the *tag* option or with a preceding call to `PDF_begin_item()`.

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

Table 7.7 Path operation options for `PDF_draw_path()` for controlling all subpaths in a path object

option	explanation
<b>bboxexpand</b>	(List of floats; will be ignored if the <i>boundingbox</i> option is specified) One or two floats which indicate the expansion of the automatically calculated bounding box (the smallest enclosing rectangle of the path object). Default: {0 0}
<b>bounding-box</b>	(Rectangle) A rectangle in the coordinate system of the path object which will be used as bounding box for fitting the path object into the fitbox. Default: the smallest enclosing rectangle of the path object, possibly expanded according to the <i>bboxexpand</i> option
<b>clip</b>	(Boolean) If <code>true</code> the path will be closed and used as clipping path. Default: <code>false</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>fill</b>	(Boolean; overrides <i>clip</i> ) If <code>true</code> each path will be filled. 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 grooved so that the corners are circular 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: the value specified when the path was constructed, or 0 if no value was specified
<b>stroke</b>	(Boolean; overrides <i>clip</i> ) If <code>true</code> the path will be stroked. Default: <code>false</code>
<b>subpaths</b>	(List of integers or single keyword) List with the numbers of subpaths to be drawn; the first subpath has number 1. The keyword <code>all</code> specifies all subpaths. Default: <code>all</code>

C++ Java C# **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:

- ▶ Keywords for querying the results of object fitting according to Table 6.3:  
*boundingbox, fitscalex, fitscaley, height, objectheight, objectwidth, width, x1, y1, x2, y2, x3, y3, x4, y4*
- ▶ Additional keywords according to Table 7.8:  
*bboxwidth, bboxheight, numpoints, px, py*

Table 7.8 Keywords for **PDF\_info\_path()**

keyword	explanation
<b>bboxwidth, bboxheight</b>	Width and height of the bounding box for the path
<b>numpoints</b>	Number of supplied points. The option <i>subpaths</i> 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 <i>subpaths</i> will be ignored.

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

- ▶ All options of **PDF\_draw\_path()** according to Table 7.7
- ▶ Additional fitting option according to Table 6.1: *refpoint*
- ▶ Additional option according to Table 7.9: *name*

**Returns** The value of some path property 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 7.9 Options for **PDF\_info\_path()**

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

---

C++ Java C# **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

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

Table 8.1 lists relevant global options for this section (see Section 2.3, »Global Options«, page 25).

Table 8.1 Color-related options for `PDF_set_option()`

option	description
<code>preserveold-pantonenames</code>	(Boolean) If false, old-style Pantone spot color names will be converted to the corresponding new color names, otherwise they will be preserved. Default: false
<code>spotcolorlookup</code>	(Boolean) If false, PDFlib does 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

---

C++ Java C# `void setcolor(String fstyle, String colorspace, double c1, double c2, double c3, double c4)`

Perl PHP `setcolor(string fstyle, string colorspace, float c1, float c2, float c3, float c4)`

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

---

Set the current color space and color for the graphics and text state.

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

**colorspace** Specifies the color space 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`).

**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_ext()` or `PDF_shading_pattern()`. The current fill or stroke color will be applied when a pattern with *painttype=uncolored* 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 *cmyk* color spaces and the *tint* value for the *spot* color space must be numbers in the inclusive range 0–1. Unused parameters should be set to 0. More information about color spaces and color values can be found in »Color options«, page 14.

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

If the *iccbasedgray/rgb/cmyk* color spaces are used, a suitable ICC profile must have been set before using one of the *iccprofilegray/rgb/cmyk* options.

This function is equivalent to `PDF_set_text_option()` and `PDF_set_graphics_option()` with the *fillcolor* and/or *strokecolor* options. `PDF_setcolor()` overrides the values of these options.

**PDF/A** *colorspace=gray* requires an output intent (any type) or the *defaultgray* option in `PDF_begin_page_ext()`.

*colorspace=rgb* requires an RGB output intent or the *defaultrgb* option in `PDF_begin_page_ext()`.

*colorspace=cmyk* requires a CMYK output intent or the *defaultcmyk* option in `PDF_begin_page_ext()`.

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

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

PDF/X-3/4/5: *colorspace=gray* requires a grayscale or CMYK device output intent or the *defaultgray* option in `PDF_begin_page_ext()`.

*colorspace=rgb* requires an RGB output intent or the *defaultrgb* option in `PDF_begin_page_ext()`.

*colorspace=cmyk* requires a CMYK output intent or the *defaultcmyk* option in `PDF_begin_page_ext()`.

**PDF/UA** Information should not be conveyed by color or contrast alone.

**Scope** *page*, *pattern* (only if *painttype=colored*), *template*, *glyph* (only if the Type 3 font's *colored* 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()`.

---

C++ Java C# ***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.

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()* or the *fillcolor* and *strokecolor* options of *PDF\_set\_graphics\_option()* and other functions. 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 with PANTONE and HKS colors, and the *spotcolorlookup* option is *true* (which is default), the specified spot color name and corresponding internal alternate color values are used. Otherwise the color values of the current fill color are 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 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.

**PDF/X** PANTONE® colors are not supported in PDF/X-1a mode.

**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.

## 8.2 ICC Profiles

C++ Java C# `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.

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

**optlist** An option list describing aspects of profile handling:

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ 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-1a/3 and <i>usage=outputintent</i> ; will be forced to true for PDF/X-4 and PDF/X-5g) Embed the ICC profile even if a standard output intent for PDF/X-1a/3 has been supplied as <i>profilename</i> . Default: false
<b>metadata</b>	(Option list; ignored for <i>usage=outputintent</i> in PDF/X-4p and PDF/X-5pg) Supply metadata for the profile (see Section 14.2, »XMP Metadata«, page 257)
<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 as ICC-based color space for text or graphics, will be applied to an image, used as default color space or as blending color space for a transparency group. <b>outputintent</b> The ICC profile specifies a PDF/A or PDF/X output intent.

**Returns** A profile handle which can be used in subsequent calls to `PDF_load_image()` or for setting profile-related options. 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 reused across multiple PDF documents. 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. Depending on the intended use, ICC profiles must satisfy the conditions listed in the PDFlib Tutorial. The *sRGB* profile is always available internally, and must not be configured.



*PDF/A* The output intent can be set using this function or by copying an imported document's output intent using *PDF\_process\_pdi()*. If only device-independent colors are used in the document no output intent is required.

*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-1/3: One of the following standard output condition names can be used without embedding the corresponding ICC profile:

CGATS TR 001, CGATS TR 002, CGATS TR 003, CGATS TR 005, CGATS TR 006,  
FOGRA30, FOGRA31, FOGRA32, FOGRA33, FOGRA34, FOGRA35, FOGRA36, FOGRA38, FOGRA39  
FOGRA40, FOGRA41, FOGRA42, FOGRA43, FOGRA44, FOGRA45, FOGRA46, FOGRA47,  
IFRA26, IFRA30,  
EUROSB104, EUROSB204,  
JC200103, JC200104, JCN2002, JCW2003

PDF/X-4: an output intent profile must be available when generating the PDF and will be embedded.

PDF/X-4/5: a CMYK output intent profile (i.e. loaded with *usage=outputintent*) can not be used for an ICCBased color space (i.e. loaded with *usage=iccbased*) 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 profiles. If you have a requirement to use the same CMYK ICC profile as in the output intent also as ICCBased color (e.g. for tagging an image), you can simply omit the ICC profile since PDF/X implies that the output intent profile will be used anyway.

PDF/X-4p/5pg: The profile is not embedded, but a reference to an external profile will be created. The profile must be available when generating the PDF, and must be available to the PDF consumer when viewing or printing the document. The PDFlib Tutorial contains an important note about Acrobat problems with referenced ICC profiles.

If one of the standard output intents listed above is to be used with PDF/X-4 or PDF/X-5, the corresponding ICC profile must be configured as with the *ICCProfile* resource.

*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*.

## 8.3 Patterns and Shadings

C++ Java C# `int begin_pattern_ext(double width, double height, string optlist)`

Perl PHP `int begin_pattern_ext(float width, float height, string optlist)`

C `int PDF_begin_pattern_ext(PDF *p, double width, double height, const char *optlist)`

Start a pattern definition with options.

**width, height** The dimensions of the pattern's bounding box in the pattern coordinate system.

**optlist** An option list describing pattern details according to Table 8.3:  
*boundingbox, painttype, tilingtype, topdown, transform, xstep, ystep*

**Returns** A pattern handle that can be used in subsequent calls to `PDF_setcolor()` and for the options *fillcolor* and *strokecolor* during the enclosing *document* scope.

**Details** This function resets all text, graphics, and color state parameters to their default values. The *transform* option defines the mapping of the pattern coordinate system to the coordinate system of the page, template or glyph description where the pattern is used. All text and graphics operations can be used on a pattern description, but hypertext functions and functions for opening images, PDF pages or graphics must be avoided. Color operations can be used depending on the *painttype* option.

**Bindings** any except *object*; this function starts *pattern* scope, and must always be paired with a matching `PDF_end_pattern()` call.

Table 8.3 Options for `PDF_begin_pattern_ext()`

Option	explanation
<b>boundingbox</b>	(Rectangle) Coordinates of the left, bottom, right, and top edges of the pattern cell's bounding box. The bounding box can be used to clip the pattern cell or to create white space around the visible pattern elements. Default: {0 0 width height}
<b>painttype</b>	(Keyword) 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 (default: colored): <b>colored</b> (Equivalent to <code>painttype=1</code> in the deprecated function <code>PDF_begin_pattern()</code> ) The pattern is colored with one or more calls to <code>PDF_setcolor()</code> or the options <code>fillcolor/strokecolor</code> . The pattern description may place images, PDF pages or graphics. <b>uncolored</b> (Equivalent to <code>painttype=2</code> in the deprecated function <code>PDF_begin_pattern()</code> ) 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, but not any images, placed PDF pages or graphics. Before using the pattern, <code>PDF_setcolor()</code> or the options <code>fillcolor/strokecolor</code> must be called to set the current color with a color space which is not based on a pattern.

Table 8.3 Options for `PDF_begin_pattern_ext()`

Option	explanation
<b>tilingtype</b>	<p>(Keyword) Controls adjustments to the spacing of pattern tiles (default: constantspacing):</p> <p><b>constantspacing</b> Pattern cells will be spaced consistently, i.e. by a multiple of a device pixel. The PDF consumer may need to distort the pattern cell slightly by making small adjustments to <code>xstep</code>, <code>ystep</code>, and the transformation matrix.</p> <p><b>nodistortion</b> The pattern cell will not be distorted, but the spacing between pattern cells may vary by as much as one device pixel, both horizontally and vertically, when the pattern is painted. This achieves the requested spacing on average but not necessarily for each individual pattern cell.</p> <p><b>fastertiling</b> Pattern cells will be spaced consistently as with constanttiling but with additional distortion permitted to enable a more efficient implementation.</p>
<b>topdown</b>	<p>(Boolean) If <code>true</code>, the origin of the coordinate system at the beginning of the pattern definition is assumed in the top left corner of the pattern bounding box and <code>y</code> coordinates increase downwards; otherwise the pattern coordinate system is used directly. Default: <code>false</code></p>
<b>transform</b>	<p>(Transformation list) A list which defines the transformation matrix that maps the pattern coordinate system to the default coordinate system of the page, template or glyph description where the pattern is used. The concatenation of the pattern matrix with that of the page, template or glyph description establishes the pattern coordinate system, within which all graphics objects in the pattern will be interpreted.</p> <p>The list contains pairs of a keyword and a float list according to in Table 8.3. where each pair defines a transformation. The interpretation and length of the number list depends on the transformation. The transformations are applied in the specified order. The elements within a pair may be separated with equals signs '='. By default no transformations are applied.</p> <p>Example: <code>transform={rotate=45 translate={100 0}}</code></p>
<b>xstep</b>	<p>(Float) Horizontal spacing between pattern cells in pattern coordinates. Default: <code>width</code></p>
<b>ystep</b>	<p>(Float) Vertical spacing between pattern cells in pattern coordinates. Default: <code>height</code></p>

Table 8.4 Keywords and Float lists for the `transform` option of `PDF_begin_pattern_ext()`

Option	explanation
<b>align</b>	<p>Rotate by the direction vector <code>{dx dy}</code>.</p>
<b>matrix</b>	<p>Specify a transformation matrix by its six components <code>{a b c d e f}</code>.</p>
<b>rotate</b>	<p>Rotate by <code>{phi}</code>, where the angle <code>phi</code> is measured in degrees counterclockwise from the positive <code>x</code> axis of the pattern coordinate system.</p>
<b>scale</b>	<p>Scale by <code>{sx sy}</code>. If <code>sy</code> is not provided it is assumed to be equal to <code>sx</code>.</p>
<b>skew</b>	<p>Skew (shear) by <code>{alpha beta}</code>, where <code>alpha</code> is measured in degrees counterclockwise from the positive <code>x</code> axis of the pattern coordinate system and <code>beta</code> is measured clockwise from the positive <code>y</code> axis. Both angles must be in the range <math>-360^\circ &lt; \alpha, \beta &lt; 360^\circ</math>, and must be different from <math>-270^\circ, -90^\circ, 90^\circ</math>, and <math>270^\circ</math>.</p>
<b>translate</b>	<p>Translate by <code>{tx ty}</code></p>

---

C++ Java C# **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\_ext()* call.

---

C++ Java C# **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)**

---

Deprecated, use *PDF\_begin\_pattern\_ext()*

---

C++ Java C# **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.

**shading** A shading handle returned by *PDF\_shading()*.

**optlist** An option list describing the graphics appearance of the shading pattern according to Table 7.1: *gstate*

*Returns* A pattern handle that can be used in subsequent calls to *PDF\_setcolor()* and for the options *fillcolor* and *strokecolor* 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()* or the options *fillcolor* and *strokecolor* to set the current color to the shading pattern.

*Scope* any except *object*

---

C++ Java C# **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.

**shading** A shading handle returned by *PDF\_shading()*.

*Details* This function allows shadings to be used without involving *PDF\_shading\_pattern()* and *PDF\_setcolor()* or the options *fillcolor* and *strokecolor*. 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* op-

tions of the shading) this function will generally be used in combination with `PDF_clip()`.

*Scope* `page`, `pattern` (only if `painttype=colored`), `template`, `glyph` (only if the Type 3 font's `colored` option is `true`)

---

C++ Java C# `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.

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

**xo, yo, x1, y1** 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.

**c1, c2, c3, c4** Color values of the shading's endpoint, interpreted in the current fill color space in the same way as the color parameters in `PDF_setcolor()` and the options `fillcolor` and `strokecolor`. If the current fill color space is a spot color space `c1` will be ignored, and `c2` contains the tint value.

**optlist** An option list describing aspects of the shading according to Table 8.5. 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 is used as the starting color; it must not be based on a pattern.

*Scope* any except `object`

Table 8.5 Options for `PDF_shading()`

Option	explanation
<b>antialias</b>	(Boolean) Specifies whether to activate antialiasing for the shading. Default: false
<b>boundingbox</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>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>extendo</b>	(Boolean) Specifies whether to extend the shading beyond the starting point. Default: false
<b>extend1</b>	(Boolean) Specifies whether to extend the shading beyond the endpoint. Default: false
<b>N</b>	(Float) Exponent for the color transition function; must be $> 0$ . Default: 1

Table 8.5 Options for `PDF_shading()`

<b>Option</b>	<b>explanation</b>
<b><i>ro</i></b>	<i>(Float; only for radial shadings, and required in this case) Radius of the starting circle</i>
<b><i>ri</i></b>	<i>(Float; only for radial shadings, and required in this case) Radius of the ending circle</i>
<b><i>startcolor</i></b>	<i>(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</i>

# 9 Image, SVG and Template Functions

## 9.1 Images

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

---

```
C++ Java C# 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 (not possible for CCITT and raw images). Explicitly specifying the image format with one of the strings *bmp*, *ccitt*, *gif*, *jbig2*, *jpeg*, *jpeg2000* (PDF 1.5 and above), *png*, *raw*, or *tiff* offers slight performance advantages. Details of all image formats are discussed in the PDFlib Tutorial.

**filename** (Name string; will be interpreted according to the global *filenamehandling* option, see Table 2.3) 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 tries to determine the appropriate file name suffix automatically; it appends all suffixes from the following list (in both lowercase and uppercase) to the specified *filename* and tries 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, .jp2, .jpf, .jpm,
.png, .raw, .tif, .tiff
```

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

**optlist** An option list specifying image-related properties according to Table 9.1. The following options can be used:

- ▶ General options: *errorpolicy* (see Table 2.1) and *hypertextencoding* (see Table 2.3)
- ▶ Color-related options: *colorize*, *honoriccprofile*, *iccprofile*, *invert*, *renderingintent*
- ▶ Clipping, masking, and transparency options:  
*alphachannelname*, *clippingpathname*, *downsamplemask*, *honorclippingpath*, *ignoremask*, *mask*, *masked*
- ▶ Special PDF features for using the image: *createtemplate*, *interpolate*
- ▶ The following common XObject options can be used (see Table 9.10):  
*associatedfiles*, *georeference*, *iconname*, *layer*, *metadata*, *OPI-1.3*, *OPI-2.0*, *pdfvt*, *reference*
- ▶ Option for analyzing the image without writing PDF output: *infomode*
- ▶ Options for processing the image data:  
*cascadedflate*, *ignoreorientation*, *page*, *passthrough*
- ▶ Options for CCITT, JBIG2 and raw images according to Table 9.2:  
*bitreverse*, *bpc*, *components*, *copyglobals*, *height*, *imagehandle*, *inline*, *K*, *width*

**Returns** An image handle (or template handle if *createtemplate=true*) 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. If the application cannot prevent this situation you can remove redundant image data with the *optimize* option of *PDF\_begin\_document()*.

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 with *PDF\_fit\_image()*, the actual image file will not be kept open after this call.

**PDF/A** Some options are restricted.

Grayscale images require an output intent (any type) or the *defaultgray* option in *PDF\_begin\_page\_ext()*.

RGB images require an RGB output intent or the *defaultrgb* option in *PDF\_begin\_page\_ext()*.

CMYK images require a CMYK output intent or the *defaultcmyk* option in *PDF\_begin\_page\_ext()*.

PDF/A-2/3: JPEG 2000 images must satisfy certain conditions; see PDFlib Tutorial for details.

**PDF/X** Some options are restricted.

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 *defaultcmyk* option in *PDF\_begin\_page\_ext()* must have been set unless the output intent is a CMYK device.

JPEG 2000 images must satisfy certain conditions; see PDFlib Tutorial for details.

**PDF/VT** This call may fail if the *usestransparency=false* option was specified in *PDF\_begin\_document()*, but the imported image contains transparency nevertheless.

**PDF/UA** Images should be tagged as *Figure* or *Artifact*.

**Scope** any except *object*; should be paired with a matching call to *PDF\_close\_image()*.



Table 9.1 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>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; 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>create-template</b>	(Boolean) If true, generate a PDF Image XObject embedded in a Form XObject (called template in PDFlib) instead of a plain Image XObject. This can be useful for creating templates for form field icons which consist of an image only. A handle for the generated template is returned. Default: false
<b>downsample-mask</b>	(Boolean; only for <code>type=TIFF</code> and <code>PNG</code> ) If true, downsample 16-bit alpha channels to 8-bit in order to work around a bug in Acrobat 7/8/9. Setting this option to true can be used to fix incorrect display of 16-bit masks in older Acrobat versions. Default: false
<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> ; forced to false if the <code>colorize</code> option is specified) Honor an ICC profile which may be requested by the image, either directly by embedding or indirectly (e.g. via a reference in the Exif marker) and apply it to the image. Default: true
<b>iccprofile</b>	(ICC handle or keyword) Handle of an ICC profile which is applied to the image. The keyword <code>srgb</code> selects the sRGB color space. Default: an embedded profile (or equivalent Exif information in the image file) if present in the image and <code>honoriccprofile=true</code> .
<b>ignoremask</b>	(Boolean; must be set to true in PDF/X-1/3 and PDF/A-1 for images with an alpha channel) Ignore transparency information and alpha channels in the image. Default: false
<b>ignore-orientation</b>	(Boolean; only for <code>imagetype=tiff</code> and <code>jpeg</code> ) Ignores any orientation information in the image. This may be useful for compensating incorrect orientation info in the image data. Default: false
<b>infomode</b>	(Boolean) If true the image is loaded, but no pixel data is written to the output. Image properties can be queried with <code>PDF_info_image()</code> , but the image cannot be placed on a page with <code>PDF_fit_image()</code> or other functions. This option may be useful to check images without any side effects on the PDF output. If false the pixel data is written to the PDF output immediately. 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

Table 9.1 Options for `PDF_load_image()`

key	explanation
<b>mask</b>	<p>(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. Default: false. There are two uses for masks:</p> <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> <p>This option forces <code>ignoremask=true</code> since an image which is used as mask cannot itself have an internal mask.</p>
<b>masked</b>	<p>(Image handle; will be ignored if the image contains an alpha channel and <code>ignoremask=false</code>) 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/A-1 and PDF/X-1/3 this option is only allowed with 1-bit masks.</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>TIFF images (default: true): If true, compressed TIFF image data is directly passed through to the PDF output if possible. Setting this option to false may help in cases where a TIFF image contains damaged or incomplete data.</p> <p>JPEG images (default: false): If false, PDFlib transcodes JPEG image data for compatibility with Acrobat. If true, JPEG image data is directly copied to the PDF output. This option is ignored for multiscan and certain CMYK JPEG images. Setting this option to true may speed up processing, but certain rare JPEG flavors won't display correctly in Acrobat.</p>
<b>rendering-intent</b>	<p>(Keyword) Rendering intent for the image (default: Auto):</p> <p><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</p> <p><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</p> <p><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</p> <p><b>Saturation</b> Saturation of the colors will be preserved while the color values may be shifted. Typical use: business graphics</p> <p><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</p>
<b>template</b>	<p>Deprecated, use <code>createtemplate</code></p>

Table 9.2 Options for `PDF_load_image()` with `imagetype=ccitt`, `jbig2` or `raw`

key	explanation
<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, 8 or 16 (in PDF 1.4 the value 16 is not allowed)
<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: current): <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>height</b>	(Integer; only for <code>imagetype=raw</code> and <code>ccitt</code> ; required in this case) Image height in pixels.
<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>copyglobals=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> ; not allowed if one of the options <code>iccprofile</code> or <code>colorize</code> is provided) 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 supplied <code>PDF_close_image()</code> must not be called. 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>width</b>	(Integer; only for <code>imagetype=raw</code> and <code>ccitt</code> ; required in this case) Image width in pixels

C++ Java C# **void close\_image(int image)**

Perl PHP **close\_image(int image)**

C **void PDF\_close\_image(PDF \*p, int image)**

Close an image or template.

**image** A valid image or template handle retrieved with `PDF_load_image()` or `PDF_begin_template_ext()`.

**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 al-

ready 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.

*Scope* any except *object*; must always be paired with a matching call to `PDF_load_image()` (unless the *inline* option has been used) or or `PDF_begin_template_ext()`.

---

C++ Java C# **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 `PDF_load_image()` or `PDF_begin_template_ext()`. The image must not have been loaded with *infomode=true*.

**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 9.3:  
*adjustpage, gstate, ignoreclippingpath, ignoreorientation*
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

*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.

*PDF/UA* All raster images must be tagged as *Artifact* or *Figure*, either with the *tag* option or with a preceding call to `PDF_begin_item()`.

*Scope* *page, pattern* (only if the pattern's *painttype* is *colored* or the image is a mask), *template, glyph* (only if the Type 3 font's *colored* 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()`.

---

C++ Java C# **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 or template and query metrics and other properties.

**image** A valid image or template handle retrieved with `PDF_load_image()` or `PDF_begin_template_ext()`.

**keyword** A keyword specifying the requested information:

Table 9.3 Options for image, graphics, PDF page and template processing with `PDF_fit_image()`, `PDF_fit_graphics()`, `PDF_fit_pdi_page()`, `PDF_fill_imageblock()`, `PDF_fill_graphicsblock()` and `PDF_fill_pdfblock()`

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>; not for <code>PDF_fill_*block()</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 (x, y) with the function parameters x and y. The MediaBox will be adjusted, and all other box entries will be reset to their defaults. With the value o for the position option the following useful cases shall be noted:</p> <p><math>x \geq o</math> and <math>y \geq o</math>                      The object is surrounded by a white margin. This margin has thickness y in horizontal direction and thickness x in vertical direction.</p> <p><math>x &lt; o</math> and <math>y &lt; o</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 and JPEG images) Ignore any orientation information in the image. This may be useful for compensating wrong orientation info. Default: the value of the ignoreorientation option in <code>PDF_load_image()</code></p>

- ▶ Keywords for querying the results of object fitting according to Table 6.3: `boundingbox`, `fitscalex`, `fitscaley`, `height`, `objectheight`, `objectwidth`, `width`, `x1`, `y1`, `x2`, `y2`, `x3`, `y3`, `x4`, `y4`
- ▶ Additional keywords according to Table 9.4: `clippingpath`, `checkcolorspace`, `filename`, `iccprofile`, `imageheight`, `imagemask`, `imagetype`, `imagewidth`, `infomode`, `mirroringx`, `mirroringy`, `orientation`, `resx`, `resy`, `strips`, `transparent`, `xid`

**optlist** The following options are supported:

- ▶ Options for `PDF_fit_image()`. Options which are not relevant for determining the value of the requested keyword are ignored.
- ▶ Option for switching between underlying image and template: `useembeddedimage`

**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 returns a handle to the object, or -1 (in PHP: o) if the object is not available. If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with `PDF_get_string()`.

**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 any except object

Table 9.4 Keywords for `PDF_info_image()`

keyword	explanation
<code>clippingpath</code>	Path handle of the image's clipping path, or -1 (in PHP: 0) if no clipping path is present
<code>checkcolorspace</code>	1 if the image or template can safely be placed on the current page without risking a color-related violation of PDF/A or PDF/X; 0 otherwise. This check takes into account the page's default color space which is not checked when loading the image.
<code>filename</code>	String index for the name of the image file (including a searchpath directory if applicable), or -1 for templates
<code>iccprofile</code>	Handle for the ICC profile embedded in the image or -1 (in PHP: 0) if no profile is present
<code>imageheight</code>	Images: height in pixels Templates: user-supplied height, or automatically determined height for the reference option
<code>imagemask</code>	Image handle of the mask associated with the image, or -1 (in PHP: 0) if no mask is attached
<code>imagetype</code>	String index for the type (format) of the image: bmp, ccitt, gif, jbig2, jpeg, jpeg2000, png, raw, tiff for raster images. If the object related to the supplied handle has been created with <code>PDF_begin_template_ext()</code> , the string template will be returned.
<code>imagewidth</code>	Images: width in pixels Templates: user-supplied width, or automatically determined width for the reference option
<code>infomode</code>	1 if the image has been loaded with the <code>infomode</code> option, 0 otherwise
<code>mirroringx</code> , <code>mirroringy</code>	Horizontal or vertical mirroring of the image (expressed as 1 or -1) according to the supplied options
<code>orientation</code>	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.
<code>resx</code> , <code>resy</code>	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.
<code>strips</code>	Number of image strips (can be different from 1 only for certain multi-strip TIFF images)
<code>transparent</code>	1 if the image contains transparency (alpha channel with > 1 bit), otherwise 0. Transparency is assumed to be present if an alpha channel was read from the original image file, or the image has been loaded with the mask option.
<code>xid</code>	(Only for PDF/VT) String index for the GTS_XID entry of the image or template, or -1 if no GTS_XID value has been assigned. The GTS_XID string can be used in the CIP4/Summary/Content/Referenced metadata property for DPM.

Table 9.5 Option for `PDF_info_image()`

key	explanation
<code>useembedded-image</code>	(Boolean; ignored if <code>createtemplate=false</code> ) If true the information of the image embedded in the template will be queried, otherwise the information of the template. If <code>useembeddedimage=true</code> some keywords (e.g. dpi) are relevant only for the original image, but not for the generated template. In particular, the values should not be used for fitting the template created for the image. Default: false

## 9.2 SVG Graphics

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

---

C++ Java C# ***int load\_graphics(String type, String filename, String optlist)***

Perl PHP ***int load\_graphics(string type, string filename, string optlist)***

C ***int PDF\_load\_graphics(PDF \*p, const char \*type, const char \*filename, int len, const char \*optlist)***

---

Open a disk-based or virtual vector graphics file subject to various options.

**type** Type of vector graphics file. The keyword *auto* automatically determines the file type. It is equivalent to *svg* which specifies SVG graphics.

**filename** (Name string; will be interpreted according to the global *filenamehandling* global option, see Table 2.3) Name of the graphics file to be opened. This must be the name of a disk-based or virtual file; PDFlib will not pull graphics from URLs.

If a file with the specified file name cannot be found 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:

.svg, .svgz

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

**optlist** An option list specifying graphics-related properties. The following options can be used:

- ▶ General options: *errorpolicy* (see Table 2.1) and *hypertextencoding* (see Table 2.3)
- ▶ Font-related options according to Table 9.6:  
*defaultfontfamily, defaultfontoptions, fallbackfontfamily, fallbackfontoptions*
- ▶ Size options according to Table 9.6:  
*fallbackheight, fallbackwidth, forcedheight, forcedwidth*
- ▶ Image-related option according to Table 9.6: *defaultimageoptions, fallbackimage*
- ▶ Other SVG processing options according to Table 9.6: *errorconditions, lang*
- ▶ Special PDF features according to Table 9.6: *devicergb, templateoptions*

**Returns** A graphics handle which can be used in subsequent graphics-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 graphics handle can 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 vector graphics file in one of the supported formats as determined by the *type* parameter. The graphics data will be stored in memory until the graphic is closed with *PDF\_close\_graphics()* or at the end of the PDFlib object's lifetime. This function does not have any visible effect on the PDF output. In order to actually place the imported graphics somewhere in the generated document, *PDF\_fit\_graphics()* must be used. Opening the same graphics more than once per generated document is not recommended because the graphics data will be copied to the output document multiply.

PDFlib opens the graphics file with the provided *filename*, processes the contents, and closes the file before returning from this call.

Font embedding (especially relevant for PDF/A, PDF/X, and PDF/UA): font outline files for all fonts used in the graphics (or suitable default fonts) must be configured. This can be facilitated with the *enumeratefonts* option (see Section 2.3, »Global Options«, page 25).

**PDF/A** All required fonts must be embedded (see above). The *devicergb* option is only allowed if the *defaultrgb* option in *PDF\_begin\_page\_ext()* is set or the output intent is an RGB device.

PDF/A-1: graphics with transparency are not allowed.

PDF/A-2a/3a: if the graphics contain text with PUA characters, the *tag* option with the *ActualText* suboption must be provided.

**PDF/X** All required fonts must be embedded (see above).

PDF/X-1a: this function must not be called.

PDF/X-3: graphics with transparency are not allowed.

PDF/X-3/4/5: the *devicergb* option is only allowed if the *defaultrgb* option in *PDF\_begin\_page\_ext()* is set or if the output intent is an RGB device.

**PDF/VT** This call may fail if the *usestransparency=false* option was specified in *PDF\_begin\_document()*, but the imported graphics contains transparency nevertheless.

**PDF/UA** Graphics should be tagged as *Figure* or *Artifact*. All required fonts must be embedded (see above). If the graphics contain text with PUA characters, the *tag* option with the *ActualText* suboption must be provided.

**Scope** any; *object* scope is not allowed if *templateoptions* is specified

Table 9.6 Options for *PDF\_load\_graphics()*

key	explanation
<b>defaultfont-family</b>	(Name string) Name of the font family which is used if a font for some text in the graphics file is either not specified or not available. Default: Arial Unicode MS if available, otherwise Helvetica
<b>defaultfont-options</b>	(Option list) Font loading options according to Table 4.2. When a font is required for text in a graphics file and this font has not already been loaded earlier, the options specified here are supplied to <i>PDF_load_font()</i> . Default: {subsetting embedding skipembedding={latincore standardcjk} }
<b>defaultimage-options</b>	(Option list) Image loading options according to Table 9.1. When an embedded or external image is processed, the options specified here are supplied to <i>PDF_load_image()</i> . Default: { }
<b>devicergb</b>	(Boolean; in PDF/A and PDF/X-3/4/5 this option is only allowed if the <i>defaultrgb</i> option has been supplied to <i>PDF_begin_page_ext()</i> or if the output intent is an RGB device) If true, graphics and text colors in SVG are interpreted in the DeviceRGB color space instead of in sRGB, and embedded raster images are processed with <i>honoriccprofile=false</i> . Default: false



Table 9.6 Options for `PDF_load_graphics()`

key	explanation
<b>error-conditions</b>	(Option list) List of conditions which trigger an error (default: empty): <ul style="list-style-type: none"> <li><b>attributes</b> (List of strings) An error occurs if one of the specified SVG attributes is present, but is not supported in PDFlib (see PDFlib Tutorial for a list of unsupported attributes). By default, unsupported attributes are ignored.</li> <li><b>elements</b> (List of strings) An error occurs if one of the specified SVG elements is present, but is not supported in PDFlib (see PDFlib Tutorial for a list of unsupported elements). By default, unsupported elements are ignored.</li> <li><b>references</b> (List of keywords) An error occurs if one of the following types of reference cannot be resolved or executed (by default all types except <code>image</code> are silently ignored and a warning is emitted):               <ul style="list-style-type: none"> <li><b>image</b> reference to an image or graphics file; see option <code>fallbackimage</code> for default behavior</li> <li><b>internal</b> internal reference to an SVG element</li> <li><b>external</b> reference to files other than image or graphics</li> <li><b>fontfamily</b> reference to a font family</li> <li><b>font</b> reference to a full font name specified via font family, weight and style</li> </ul> </li> </ul>
<b>fallback-fontfamily</b>	(Name string) Name of the font family which is used to create a fallback font for each font, in addition to the fallback fonts which may have been specified in the graphics file. Default: empty
<b>fallback-fontoptions</b>	(Option list) Options which will be applied to the fallback fonts created via the <code>fallbackfontfamily</code> option. The following options according to Table 4.3, page 69, can be used: <code>fontsize</code> , <code>forcechars</code> , <code>textrise</code> . Default: empty
<b>fallback-height</b>	(Float; ignored if <code>forcedheight</code> is supplied) Height of the SVG graphics (in user coordinates) for the fitting process. Default: height provided in the SVG <code>viewBox</code> attribute if present, otherwise 1000
<b>fallback-image</b>	(Option list) Specify how to visualize the space reserved for the fitbox of a graphics or image element if the element is not available (colors are interpreted in sRGB if <code>devicergb=false</code> and in RGB otherwise). By default a gray semi-transparent checkerboard is painted: <ul style="list-style-type: none"> <li><b>fillcolor</b> (RGB Color or keyword) Color used to fill the area with a checkerboard pattern (if <code>gridsize &gt; 0</code>) where half of the squares are painted with the specified color and the other half of the squares is transparent, or color used to fill the area with solid color if <code>gridsize=0</code>. The keyword <code>none</code> means that the area will not be filled. Default: <code>LightGrey</code></li> <li><b>gridsize</b> (Float or percentage) Width of a square in the checkerboard pattern in default coordinates or as a percentage of the width of the fitbox. Percentages are rounded so that an integer number of squares fits into the area. <code>gridsize=0</code> means that the area is filled with solid color instead of a checkerboard pattern. Default: 10</li> <li><b>image</b> (Image or template handle) Image or template which will be placed with <code>fitmethod=entire</code> into the fitbox. Default: no image or template</li> <li><b>opacity</b> (Float in the range 0..1) Opacity of the checkerboard squares or the interior of the area. Default: 0.5</li> <li><b>strokecolor</b> (RGB color or keyword) Color used to stroke the border area and a cross inside the area. The keyword <code>none</code> means that the border and cross is not stroked. Default: <code>Red</code></li> </ul>
<b>fallback-width</b>	(Float; ignored if <code>forcedwidth</code> is supplied) Width of the SVG graphics (in user coordinates) for the fitting process. Default: width provided in the SVG <code>viewBox</code> attribute if present, otherwise 1000
<b>forcedheight</b>	(Float) The height of the SVG graphics (if present) is ignored and the specified value in the default coordinate system is applied instead. Default: height of the graphics
<b>forcedwidth</b>	(Float) The width of the SVG graphics (if present) is ignored and the specified value in the default coordinate system is applied instead. Default: width of the graphics
<b>lang</b>	(String) Natural language for the graphics file which can be used e.g. in an SVG switch element. The format of the language specification is identical to the <code>lang</code> option of <code>PDF_begin_document()</code> (see Table 3.3). Default: the language identifier found in the <code>LANG</code> environment variable.

Table 9.6 Options for `PDF_load_graphics()`

key	explanation
<b>template-options</b>	<p>(Option list) Create a template (PDF Form XObject) according to the supplied option list. This option is recommended if the graphics will be placed more than once, or if specific template features are required (e.g. for PDF/VT). The supplied option list (which may be empty) is used for <code>PDF_begin_template_ext()</code>. The following common XObject options can be used (see Table 9.10): <code>associatedfiles</code>, <code>iconname</code>, <code>layer</code>, <code>metadata</code>, <code>pdfvt</code>, <code>transparencygroup</code>.</p> <p>Width and height of the template are determined based on the size of the graphics. The template is written to the PDF output in <code>PDF_close_graphics()</code> or at the end of the document (the latter only if <code>PDF_fit_graphics()</code> was called at least once for this graphics file).</p>

C++ Java C# **void close\_graphics(int graphics)**

Perl PHP **close\_graphics(int graphics)**

C **void PDF\_close\_graphics(PDF \*p, int graphics)**

Close vector graphics.

**graphics** A valid graphics handle retrieved with `PDF_load_graphics()`.

**Details** PDFlib's associated internal graphics structure will be deleted. If the `templateoptions` option was specified in `PDF_load_graphics()` the corresponding PDF template will be created before closing the graphic. If the graphic has been opened from file, the actual graphics file is not affected by this call since it has already been closed at the end of the corresponding `PDF_load_graphics()` call. A graphics handle cannot be used any more after it has been closed with this function.

**Scope** any; *object* scope is not allowed if `templateoptions` was specified in the corresponding call to `PDF_load_graphics()` and the graphics was placed at least once; must always be paired with a matching call to `PDF_load_graphics()`.

C++ Java C# **void fit\_graphics(int graphics, double x, double y, String optlist)**

Perl PHP **fit\_graphics(int graphics, float x, float y, string optlist)**

C **void PDF\_fit\_graphics(PDF \*p, int graphics, double x, double y, const char \*optlist)**

Place vector graphics on a content stream, subject to various options.

**graphics** A valid graphics handle retrieved with `PDF_load_graphics()`.

**x, y** The coordinates of the reference point in the user coordinate system where the graphic will be placed.

**optlist** An option list specifying graphics fitting and processing options. The following options are supported:

- ▶ Fitting options according to Table 6.1:  
*blind*, *boxsize*, *fitmethod*, *matchbox*, *orientate*, *position*, *refpoint*, *rotate*, *scale*, *showborder*
- ▶ Options for graphics processing according to Table 9.3:  
*adjustpage*, *gstate*
- ▶ Option for processing interactive links in the graphics according to Table 9.7:  
*convertlinks*
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

**Details** The graphics 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, a graphic will be scaled according to its internally specified size. This behavior can be modified with the *scale* and *fitmethod* options.

It is recommended to use `PDF_info_graphics()` with the *fittingpossible* keyword before calling `PDF_fit_graphics()` to check whether `PDF_fit_graphics()` will succeed (and avoid an exception in case of a failure).

**PDF/UA** Graphics containing vector graphics or raster images must be tagged as *Figure* or *Artifact*.

**Scope** *page*, *pattern* (only if the pattern's *painttype* is 1), *template*, *glyph* (only if the Type 3 font's *colored* option is *true*); this function can be called an arbitrary number of times on arbitrary pages as long as the graphics handle has not been closed with `PDF_close_graphics()`.

Table 9.7 Additional option for `PDF_fit_graphics()`

key	explanation
<b>convertlinks</b>	(Boolean) If <code>true</code> , interactive links in the graphics file are converted to interactive Link annotations in PDF. Regardless of this setting links are not created in the following situations (default: <code>true</code> ): <ul style="list-style-type: none"> <li>▶ this function is called in a scope other than page</li> <li>▶ the <i>templateoptions</i> option was supplied in <code>PDF_load_graphics()</code></li> <li>▶ Tagged PDF mode: the currently active structure item is an <i>Artifact</i>;</li> <li>▶ PDF/X: the link annotation is located inside the <i>BleedBox</i> (or <i>TrimBox</i>/<i>ArtBox</i> if no <i>BleedBox</i> is present).</li> </ul>

C++ Java C# **double** `info_graphics(int graphics, String keyword, String optlist)`

Perl PHP **float** `info_graphics(int graphics, string keyword, string optlist)`

C **double** `PDF_info_graphics(PDF *p, int graphics, const char *keyword, const char *optlist)`

Format vector graphics and query metrics and other properties.

**graphics** A valid graphics or template handle retrieved with `PDF_load_graphics()`.

**keyword** A keyword specifying the requested information:

- ▶ Keywords for querying the results of object fitting according to Table 6.3: *boundingbox*, *fitscalex*, *fitscaley*, *height*, *objectheight*, *objectwidth*, *width*, *x1*, *y1*, *x2*, *y2*, *x3*, *y3*, *x4*, *y4*
- ▶ Additional keywords according to Table 9.8: *description*, *filename*, *fittingpossible*, *graphicswidth*, *graphicsheight*, *istemplate*, *metadata*, *title*, *type*, *xid*

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

**Returns** The value of some graphics property as requested by *keyword*. If a geometrical property is requested outside of a page, this function returns -1 (in PHP: 0). If an object handle is requested this function returns a handle to the object, or -1 (in PHP: 0) if the object is not available. If the requested keyword produces text, a string index is returned, and the corresponding string must be retrieved with `PDF_get_string()`.

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

**Scope** any

Table 9.8 Keywords for `PDF_info_graphics()`

<b>keyword</b>	<b>explanation</b>
<b>description</b>	String index for the contents of the desc element of the outermost svg element if present, or of the outermost g element if present, or -1 otherwise. The string may contain markup.
<b>filename</b>	String index for the name of the graphics file (including a searchpath directory if applicable)
<b>fittingpossible</b>	<p>Check whether the graphics could be placed with <code>PDF_fit_graphics()</code> in the current context. The value 1 is returned if the graphics can be placed. The value 0 is returned if fitting fails (i.e. <code>PDF_fit_graphics()</code> would throw an exception) for one of the following reasons:</p> <ul style="list-style-type: none"> <li>▶ An internal problem in the graphics file.</li> <li>▶ A conflict with current standards requirements (e.g. transparency not allowed, font must be embedded but no font file available)</li> </ul> <p>If 0 is returned the nature of the problem can be queried with <code>PDF_get_errmsg()</code>. Since the result is valid only for the current context this check should be applied immediately before attempting to place a page.</p>
<b>graphicswidth, graphicsheight</b>	Width and height of the graphic in the default coordinate system according to the information in the graphics file. If no values are available in the graphics file, 0 will be returned.
<b>istemplate</b>	1 if the <code>templateoptions</code> option has been supplied, 0 otherwise
<b>metadata</b>	String index for the contents of the metadata element of the outermost svg element, or -1 if this element is not present. The string may contain markup.
<b>title</b>	String index for the contents of the title element of the outermost svg element if present, or of the outermost g element if present, or -1 otherwise. The string may contain markup.
<b>type</b>	String index for the type (format) of the graphics: always <code>svg</code>
<b>xid</b>	(Only for PDF/VT) String index for the GTS_XID entry of the template created for graphic, or -1 if no template was created or no GTS_XID value has been assigned to the template. The GTS_XID string can be used in the CIP4/Summary/Content/Referenced metadata property for DPM.

## 9.3 Templates

*Note* We use the term »template« as a synonym for PDF Form XObjects. The template functions described in this section are unrelated to variable data processing with PDFlib Blocks. Use `PDF_fill_block()` to fill Blocks prepared with the PDFlib Block Plugin (see Chapter 11, »Block Filling Functions (PPS)«, page 203).

---

C++ Java C# `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 option of `PDF_begin_page_ext()` can be used (see Table 3.9): *topdown*
- ▶ The following common XObject options can be used (see Table 9.10):  
*associatedfiles, iconname, layer, metadata, OPI-1.3, OPI-2.0, pdfvt, reference, transparencygroup*
- ▶ The following transformation option can be used (see Table 8.3): *transform*
- ▶ The following option for including PostScript code can be used (see Table 9.9):  
*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 resets all text, graphics, and color state parameters to their default values, and establishes a coordinate system according to the *topdown* option. Hypertext functions must not be used during a template definition, but all text, image, 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.

**PDF/A** The *postscript* option must not be used.

**PDF/X** The *postscript* option must not be used.

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

Table 9.9 Option for `PDF_begin_template_ext()`

key	explanation
<code>postscript</code>	<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 post-processing 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 <code>template</code>, no API function calls for producing graphical output are allowed until the corresponding call to <code>PDF_end_template_ext()</code>. 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>

C++ Java C# `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.

## 9.4 Common XObject Options

PDF XObjects are created for imported images, imported vector graphics (if the *templateoptions* list was supplied) imported PDF pages, and templates. The options listed in this section apply to the following functions which can create XObjects:

- ▶ `PDF_load_image()`
- ▶ `PDF_load_graphics()` with the *templateoptions* option
- ▶ `PDF_open_pdi_page()`
- ▶ `PDF_begin_template_ext()`

The following XObject options are available (see Table 9.10 for details): *associatedfiles*, *georeference*, *iconname*, *layer*, *metadata*, *OPI-1.3*, *OPI-2.0*, *pdfvt*, *reference*, *transparencygroup*

*PDF/A* Some options are restricted.

*PDF/X* Some options are restricted. The *reference* option is relevant for PDF/X-5g/5pg.

*PDF/VT* The *pdfvt* option is relevant for PDF/VT.

Table 9.10 Common XObject options for `PDF_load_image()`, `PDF_open_pdi_page()`, and `PDF_begin_template_ext()` as well as `PDF_load_graphics()` with the *templateoptions* option

key	explanation
<b>associatedfiles</b>	(List of asset handles; only for PDF 2.0 and PDF/A-3) Asset handles for associated files according to PDF/A-3. The files must have been loaded with <code>PDF_load_asset()</code> and <i>type=attachment</i> .
<b>georeference</b>	(Option list; PDF 1.7ext3; only for <code>PDF_load_image()</code> ) Description of an earth-based coordinate system associated with the XObject to use for geospatial measuring; see Section 12.7, «Geospatial Features», page 238, for details.
<b>iconname</b>	(Hypertext string) Attach a name to the XObject so that it can be referenced via JavaScript, e.g. to use the XObject as an icon for form fields. <code>PDF_load_image()</code> : this option will be ignored if <i>inline=true</i> ; forces <i>createtemplate=true</i>
<b>layer</b>	(Layer handle; PDF 1.5) Layer to which the XObject will belong unless another layer has been activated with <code>PDF_begin_layer()</code> prior to placing the object. Calling <code>PDF_begin_layer()</code> to activate a layer before placing the XObject overrides the object's <i>layer</i> option. Call <code>PDF_end_layer()</code> before placing the object to make sure that the object's <i>layer</i> option will not be overridden.
<b>metadata</b>	(Option list) Metadata for the XObject (see Section 14.2, «XMP Metadata», page 257).
<b>OPI-1.3</b>	(Option list; not for <code>PDF_open_pdi_page()</code> ; not for PDF/A and PDF/X) Option list containing OPI 1.3 PostScript comments as option names; the following entries are required: <i>ALDImageFilename</i> (string), <i>ALDImageDimensions</i> (list of integers), <i>ALDImageCropRect</i> (rectangle with integers), <i>ALDImagePosition</i> (list of floats) The suboption <i>normalizefilename</i> controls the handling of file names: if <i>true</i> , file names will be normalized as mandated by the PDF reference. If <i>false</i> 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: <i>false</i>
<b>OPI-2.0</b>	(Option list; not for <code>PDF_open_pdi_page()</code> ; not for PDF/A and PDF/X) Option list containing OPI 2.0 PostScript comments as option names; the following entry is required: <i>ImageFilename</i> (string) The following entries should either both be present or absent: <i>ImageCropRect</i> (rectangle), <i>ImageDimensions</i> (list of floats) The suboption <i>normalizefilename</i> is also supported (see OPI-1.3).
<b>pdfvt</b>	(Option list; only for PDF/VT) PDF/VT suboptions for the XObject according to Table 9.12

Table 9.10 Common XObject options for `PDF_load_image()`, `PDF_open_pdi_page()`, and `PDF_begin_template_ext()` as well as `PDF_load_graphics()` with the `templateoptions` option

key	explanation
<b>reference</b>	<p>(Option list; not for <code>PDF_load_image()</code>); requires Acrobat 9 or above for proper page rendering; not allowed in PDF/A, PDF/X-1/2/3/4, PDF/VT-1, and PDF/UA; not available in PDFlib source code packages) Reference a page in an external PDF (the »target« document). The page opened with <code>PDF_open_pdi_page()</code> or the template created with <code>PDF_begin_template_ext()</code> or the graphics loaded with <code>PDF_load_graphics()</code> will be used as 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 9.11 for available suboptions.</p> <p><b>PDF_open_pdi_page()</b>: The target PDF must be available locally and must contain the page addressed with the <code>pageLabel</code> or <code>pagenumber</code> suboption. The target must not require a user or master password. The size of the reference page will be determined according to the <code>pdiusebox</code> suboption of the reference option. It can be retrieved with the <code>imagewidth/imageheight</code> keywords of <code>PDF_info_image()</code>. The proxy page and the target page must have compatible page geometry, i.e. the page boxes selected with the <code>pdiusebox</code> option must be identical to make sure that both pages will be placed at the same location on the page.</p> <p><b>PDF_begin_template_ext()</b>: If <code>width</code> and <code>height</code> have been supplied with the value 0 the template size can be retrieved with the <code>imagewidth/imageheight</code> keywords of <code>PDF_info_image()</code>. If <code>width</code> and <code>height</code> have been supplied with values different from 0, the following suboptions can also be used (see Table 6.1): <code>fitmethod</code>, <code>position</code>.</p> <p><b>PDF_load_graphics()</b>: the graphic will be adjusted to the size of the target page; the following suboptions can also be used (see Table 6.1): <code>fitmethod</code>, <code>position</code>.</p> <p><b>PDF/X-5g/5pg</b>: 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> <p><b>PDF/VT-2</b>: the target must conform to one of the following standards: PDF/X-1a:2003, PDF/X-3:2003, PDF/X-4, PDF/X-4p, or PDF/VT-1 and must have been prepared for the same output intent.</p> <p>See the PDFlib Tutorial regarding the required Acrobat configuration for viewing referenced pages.</p>
<b>transparency group</b>	<p>(Option list or keyword; not for <code>PDF_load_image()</code>); not for PDF/A-1 and PDF/X-1/3; restrictions apply to PDF/A-2/3 and PDF/X-4/5) Create a transparency group for the imported page or graphics or the generated template. The following keywords are supported (default: auto):</p> <p><b>auto</b> If an imported page contains a transparency group it is copied to the generated Form XObject. However, if it conflicts with a CMYK ICC profile in a PDF/X-4 output intent it is ignored (in this case a transparency group isn't required anyway). For graphics and templates no transparency group is created.</p> <p><b>none</b> Don't create any transparency group, even for imported pages which already contain one.</p> <p>The following suboptions can be used to explicitly create a transparency group:</p> <p><b>colorspace</b> (Keyword or ICC profile handle; required with a value different from none if the template is used for the <code>softmask</code> option of <code>PDF_create_gstate()</code> with <code>type=luminosity</code>) Blending color space (default: none):</p> <p><b>DeviceCMYK</b> PDF/A-2/3 and PDF/X-4/5: only allowed with a CMYK output intent</p> <p><b>DeviceGray</b> PDF/A-2/3 and PDF/X-4/5: only allowed with a gray or CMYK output intent</p> <p><b>DeviceRGB</b> PDF/A-2/3 and PDF/X-4/5: only allowed with an RGB output intent</p> <p><b>none</b> No color space is emitted for the transparency group.</p> <p><b>srgb</b> Keyword for selecting the sRGB color space</p> <p><b>isolated</b> (Boolean) Specifies whether the transparency group is isolated. Default: false</p> <p><b>knockout</b> (Boolean) Specifies whether the transparency group is a knockout group. Default: false</p>



Table 9.11 Suboptions for the reference option in `PDF_begin_template_ext()`, `PDF_open_pdi_page()` as well as `PDF_load_graphics()` with the `templateoptions` option

key	explanation
<b>filename</b>	(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 <code>target</code> option has been supplied. It is recommended to use plain base names without any directories.
<b>hypertext-encoding</b>	(Keyword) Specifies the encoding for the <code>pagelabel</code> option. An empty string is equivalent to <code>unicode</code> . Default: value of the global <code>hypertextencoding</code> option
<b>pagelabel</b>	(Hypertext string; must not be combined with <code>pagenumber</code> ) Page label of the page to be referenced
<b>pagenumber</b>	(Integer) Number of the page to be referenced. The first page has page number 1. Default: 1 (this may be overwritten by <code>pagelabel</code> , however).
<b>pdiusebox</b>	(Keyword; will be forced to <code>media</code> in PDF/X-5g/5pg) Specifies which box dimensions will be used for determining the size of the target page. Default: <code>media</code> in PDF/X-5g/5pg mode, else <code>crop</code> . <ul style="list-style-type: none"> <li><b>media</b> Use the <code>MediaBox</code> (which is always present)</li> <li><b>crop</b> Use the <code>CropBox</code> if present, else the <code>MediaBox</code></li> <li><b>bleed</b> Use the <code>BleedBox</code> if present, else the <code>CropBox</code></li> <li><b>trim</b> Use the <code>TrimBox</code> if present, else the <code>CropBox</code></li> <li><b>art</b> Use the <code>ArtBox</code> if present, else the <code>CropBox</code></li> </ul>
<b>strongref</b>	(Boolean; will be forced to <code>true</code> in PDF/X-5g/5pg) 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) 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 <code>password</code> option. Supplying a document handle in addition to the <code>filename</code> 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>

Table 9.12 Suboptions for the pdfvt option in `PDF_load_image()`, `PDF_open_pdi_page()`, `PDF_begin_template_ext()` as well as `PDF_load_graphics()` with the `templateoptions` option

key	explanation
<b>environment</b>	(Hypertext string; required if <code>scope=stream</code> or <code>scope=global</code> ) Specifies a PDF/VT environment context, i.e. an identifier that can be used by a PDF/VT processor to provide a management interface for managing related XObjects. For example, the customer name or job name could be used to identify the environment.
<b>scope</b>	(Keyword) PDF/VT scope (not related to PDFlib function scope) of the XObject (default: unknown): <ul style="list-style-type: none"> <li><b>unknown</b> The scope of the XObject is not known.</li> <li><b>singleuse</b> The XObject is referenced only once in the PDF/VT file.</li> <li><b>record</b> (Only allowed if the <code>recordlevel</code> option for <code>PDF_begin_document()</code> has been specified) The XObject is referenced more than once in the pages belonging to a single record, but is not referenced within other records.</li> <li><b>file</b> The XObject is referenced more than once in the PDF/VT file. If the <code>recordlevel</code> option has been supplied, <code>scope=file</code> should only be used if the XObject is used in more than one record (and <code>scope=record</code> otherwise).</li> <li><b>stream</b> (Requires the <code>environment</code> option; only allowed for documents which will be included in a PDF/VT-2s stream) The XObject or an equivalent XObject is referenced more than once in the PDF/VT-2s stream containing the PDF/VT file.</li> <li><b>global</b> (Requires the <code>environment</code> option) The XObject or an equivalent XObject is referenced in more than one PDF/VT file or PDF/VT-2s stream.</li> </ul>
<b>xid</b>	(String; only for <code>PDF_begin_template_ext()</code> since identifiers will automatically be created for all other types of XObjects) Unique identifier for the Form XObject created for the template. It is strongly recommended to provide this identifier in the format suggested in ISO 16612-2:2010, section 6.7.2, i.e. a URI with the <code>uuid</code> scheme and 128-bit number according to RFC 4122. The identifiers should be identical for template definitions which create equivalent PDF Form XObjects according to PDF/VT (i.e. templates which create the same visual output). Templates which are not equivalent must have different identifiers or no identifier at all.  It is strongly recommended to supply the <code>xid</code> option for templates with <code>scope=stream</code> or <code>scope=global</code> to allow caching of Form XObjects across documents.  Example <code>xid</code> in the recommended format: <code>uuid:1228c416-48f2-e817-ad69-8206e41dca2d</code>

# 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`.

---

C++ Java C# **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, see Table 2.3) Name of the PDF file.

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

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ PDF document options according to Table 10.1:  
*checkoutintentprofile, infomode, inmemory, passwordrepair, requiredmode, shrug*
- ▶ Tagged PDF processing options according to Table 10.1:  
*checktags, , usetags*
- ▶ Layer processing options according to Table 10.1:  
*parentlayer, parenttitle, uselayers*

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

**Returns** A PDI document handle which can be used for processing individual pages of the document or for querying document properties. A return value of -1 (in PHP: 0) indicates that the PDF document couldn't be opened. An arbitrary number of PDF documents can be opened simultaneously. The return value can be used until the end of the enclosing *document* scope. If the function call fails you can request the reason of the failure with *PDF\_get\_errmsg()*.

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

**Details** By default, the document is 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 master password has not been supplied in the *password* option. The *shrug* option can be used to enable page import from protected documents under certain conditions (see PDFlib Tutorial).

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, document info fields, etc.

In order to get more detailed information about the nature of a PDF import-related problem (wrong PDF file name, bad PDF data, etc.), use `PDF_get_errmsg()` to receive a more detailed error message.

*Scope* any; in *object* scope a PDI document handle can only be used in the `PDF_pcos_get_*`(*)* functions.

Table 10.1 Options for `PDF_open_pdi_document()`

key	explanation
<b>checktags</b>	(Keyword) Specifies whether the structure element nesting rules (see PDFlib Tutorial) are checked in <code>PDF_open_pdi_page()</code> for imported structure elements. Supported keywords (default: none): <b>none</b> Tag nesting rules are not enforced. This setting is default since many real-world documents violate structure element nesting rules, and couldn't otherwise be imported. <b>relaxed</b> Similar to strict except that a few rules are not enforced (see PDFlib Tutorial). <b>strict</b> If an imported tag violates the nesting rules the call to <code>PDF_open_pdi_page()</code> will fail.
<b>checkoutintentprofile</b>	(Boolean, only relevant for PDF/A and PDF/X) If true, the number of color components of an output intent is checked against the number of components in the associated ICC profile. This guards against inconsistent input documents. Setting this option to false reduces memory requirements, but should be used only if the input documents are known to contain consistent output intent profiles. Default: true
<b>infomode</b>	(Boolean) If true, the document will be opened such that information can be queried with the pCOS interface, but the pages can not be imported into the current output document with <code>PDF_open_pdi_page()</code> . The following documents can be opened when <code>infomode=true</code> : encrypted PDFs where the password is not known (exception: PDF 1.6 and above documents created with the Distiller setting »Object Level Compression: Maximum«). Default: false if <code>requiredmode=full</code> , otherwise true
<b>inmemory</b>	(Boolean) If true, PDI will load the complete file into memory and process it from there. This can result in a tremendous performance gain on some systems (especially z/OS) at the expense of memory usage. If false, individual parts of the document will be read from disk as needed. Default: false
<b>parentlayer</b>	(Layer handle; ignored if the input document doesn't contain any layers or <code>uselayers=false</code> ) Insert all layer definitions imported from the document as children of the specified layer. If the specified layer has been activated anywhere in the output document it is used as parent; otherwise it is used as title (separator) only. Default: no parent layer
<b>parenttitle</b>	(Hypertext string; ignored if the input document doesn't contain any layers or <code>uselayers=false</code> ) Add a title layer which does not directly control the visibility of page contents, but serves as a hierarchical separator for the imported layer definitions. Default: no parent title
<b>password</b>	(String) Master password required to open a protected PDF document for import. If <code>infomode=true</code> 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. The shrug option can be used to import pages from protected documents under certain conditions (see PDFlib Tutorial).
<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: auto): <b>auto</b> Repair the document only if problems are detected while opening the PDF. <b>force</b> Unconditionally try to repair the document, regardless of whether or not it has problems. <b>none</b> No attempt will be made at repairing the document. If there are problems in the PDF the function call will fail.

Table 10.1 Options for `PDF_open_pdi_document()`

key	explanation
<code>requiredmode</code>	(Keyword) The minimum <i>pcos</i> mode (minimum/restricted/full) 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. <code>requiredmode=minimum</code> for an unencrypted document will result in full mode. Default: full
<code>shrug</code>	(Boolean) If true, the shrug feature will be activated to enable page import from protected documents under certain conditions (see PDFlib Tutorial). By using the shrug option you assert that you honor the PDF document author's rights. Default: false
<code>uselayers</code>	(Boolean; only relevant if the input contains layers) If true, all layer definitions used on any of the imported pages are imported. This option affects only layer definitions, but not the actual layer contents since PDI always imports the contents of all layers on a page. Default: true
<code>usetags</code>	(Boolean; only relevant for tagged PDF input; must be true in PDF/UA mode) If true, the structure hierarchy of the imported document is read so that structure element tags can later be imported along with the pages. Default: true

---

```
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; all options of `PDF_open_pdi_document()` are supported.

**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* any; 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 C# **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 C# *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 2.1) and *hypertextencoding* (see Table 2.3)
- ▶ PDF page options according to Table 10.2:  
*boxexpand*, *checktransgroupprofile*, *clippingarea*, *cloneboxes*, *forcebox*, *pdiusebox*, *usetags*
- ▶ Common XObject options (see Table 9.10):  
*associatedfiles*, *iconname*, *layer*, *metadata*, *pdfvt*, *reference*, *transparencygroup*

**Returns** A PDI 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 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\_open\_pdi\_page()*.

The error behavior can be changed with the *errorpolicy* 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 in the following cases:

- ▶ 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. PDF 1.7, PDF 1.7ext3, PDF 1.7ext8 and PDF 2.0 are all compatible to each other as far as page import with PDI is concerned. However, in PDF/A mode the input PDF version number is ignored since PDF version headers must be ignored in PDF/A.
- ▶ The document is not compatible to the current PDF/A, PDF/X, PDF/VT or PDF/UA output conformance level, or uses an incompatible output intent.
- ▶ If the document contains an inconsistent PDF/A or PDF/X output intent no pages can be imported.

In order to get more detailed information about a problem related to PDF import (bad PDF data, etc.) you can call *PDF\_get\_errmsg()*.

If the imported page contains referenced XObjects, *PDF\_open\_pdi\_page()* copies 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** The imported document must be compatible to the current PDF/A output conformance level (see PDFlib Tutorial for details) and output intent.
- PDF/X** The imported document must be compatible to the current PDF/X output conformance level (see PDFlib Tutorial for details), and must use the same output intent as 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.
- PDF/VT** The imported document must be compatible to the current PDF/VT output level (see PDFlib Tutorial for details), and must use the same output intent as the generated document. Document Part Metadata (DPM) in the imported document is not imported. This call may fail if the *usestransparency=false* option was specified in *PDF\_begin\_document()*, but the imported page contains transparency nevertheless.
- PDF/UA** The imported document must conform to PDF/UA. The rolemap of the imported document must be compatible with the mapping provided by the *rolemap* option of *PDF\_begin\_document()* (see PDFlib Tutorial for details). This means that custom element types must not be mapped to different standard types by the *rolemap* option (or previously imported documents) and the rolemap of the imported document.  
 The heading structure of the imported page must be compatible with the structure type of the generated document, i.e. if *structuretype=weak* only *H1*, *H2*, etc. (but not *H*) must be used on the page; if *structuretype=strong* only *H* (but not *H1*, *H2*, etc.) must be used on the imported page. Pages with both numbered and unnumbered headings are rejected.

*Scope* any except *object*

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

key	explanation
<b>boxexpand</b>	(Float or list with four floats) Expand the page box selected via the <i>pdiusebox</i> option on all four sides by the same amount (if one value is provided) or on the left/right/bottom/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>checktransgroupprofile</b>	(Boolean, only relevant for PDF/A and PDF/X) If true and the imported page contains a transparency group, its color space is checked for consistency and compatibility with the generated output document. This guards against inconsistent input documents and color space conflicts which could lead to non-conforming PDF/X or PDF/A output. Setting this option to false reduces memory requirements, but should be used only if the imported page is known to contain a conforming transparency group (if any). Default: true
<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: <i>pdiusebox</i> ): <b>art</b> Use the <i>ArtBox</i> if present, else the <i>CropBox</i> <b>bleed</b> Use the <i>BleedBox</i> if present, else the <i>CropBox</i> <b>crop</b> Use the <i>CropBox</i> if present, else the <i>MediaBox</i> <b>media</b> Use the <i>MediaBox</i> (which is always present) <b>pdiusebox</b> Use the box specified in the <i>pdiusebox</i> option <b>trim</b> Use the <i>TrimBox</i> if present, else the <i>CropBox</i>



Table 10.2 Options for `PDF_open_pdi_page()`

key	explanation
<b>cloneboxes</b>	(Boolean; not allowed if <code>boxexpand</code> , <code>forcebox</code> , or <code>pdiusebox</code> is supplied; must match the <code>cloneboxes</code> option in <code>PDF_fit_pdi_page()</code> ) If true, the page will be prepared for box cloning with the <code>cloneboxes</code> option of <code>PDF_fit_pdi_page()</code> . Default: false
<b>forcebox</b>	(Rectangle) Force the page box to the specified values. This option overrides the <code>pdiusebox</code> and <code>boxexpand</code> 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 <code>boxexpand</code> option is preferable since it works regardless of any /Rotate key. Default: the box selected with the <code>pdiusebox</code> option
<b>ignore-pdfversion</b>	(Boolean) If true, the PDF version number of the input PDF document is ignored, i.e. pages from documents with a higher PDF version than the current PDF output document can be imported. This may be useful for PDF documents with a higher PDF version which are nevertheless fully compatible to PDF 1.7. The user is responsible for making sure that the imported pages do not violate the PDF output compatibility level. Default: false in general, but true in PDF/A and PDF/X mode
<b>pdiusebox</b>	(Keyword; not allowed if <code>cloneboxes</code> 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 <code>PDF_fit_pdi_page()</code> . This box will also determine the visible contents of the page unless modified with the <code>clippingarea</code> option. Default: crop. <b>art</b> Use the ArtBox if present, else the CropBox <b>bleed</b> Use the BleedBox if present, else the CropBox <b>crop</b> Use the CropBox if present, else the MediaBox <b>media</b> Use the MediaBox (which is always present) <b>trim</b> Use the TrimBox if present, else the CropBox
<b>usetags</b>	(Boolean; only relevant for tagged PDF input and output and if the document has been opened with <code>usetags=true</code> ) If true, the structure tags of the imported page will be copied to the structure hierarchy of the generated output document. In this case <code>PDF_fit_pdi_page()</code> can only be called in page scope. Default: true

C++ Java C# **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** any except *object*

C++ Java C# **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 9.3: *adjustpage, gstate*
- ▶ The *cloneboxes* option according to Table 10.3.
- ▶ Option for abbreviated structure element tagging according to Table 14.5 (only allowed in *page* scope): *tag*

**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}`.

A tagged page (i.e. tagged PDF is created and the page is imported with `usetags=true` from a tagged PDF) cannot be placed more than once.

In Tagged PDF mode it is recommended to use `PDF_info_pdi_page()` with the `fitting-possible` keyword before calling `PDF_fit_pdi_page()` to check whether `PDF_fit_pdi_page()` will succeed (and avoid an exception in case of a failure).

**Scope** *page, pattern, template, glyph*; however, if a page from a Tagged PDF document has been loaded with `usetags=true` this function can only be called in *page* scope.

Table 10.3 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 <i>page</i> scope).</p> <p>Setting this option to <code>true</code> has the following consequences (Default: <code>false</code>):</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>

---

C++ Java C# **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:

- ▶ Keywords for querying the results of object fitting according to Table 6.3: *boundingbox, fitscalex, fitscaley, height, objectheight, objectwidth, width, x1, y1, x2, y2, x3, y3, x4, y4*
- ▶ Page-related keywords according to Table 10.4: *mirroringx, mirroringy, pageheight, pagewidth, rotate, xid*
- ▶ Keywords related to Tagged PDF according to Table 10.4: *fittingpossible, lang, topleveltag, topleveltagcount*

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

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ Fitting options according to Table 6.1 (if the PDF page has been opened with the *cloneboxes* option of *PDF\_open\_pdi\_page*() these options will be ignored): *boxsize, fitmethod, matchbox, orientate, position, rotate, scale*
- ▶ Options for page processing according to Table 9.3 don't make sense; however, they can be supplied but are ignored to facilitate unified option lists for *PDF\_fit\_pdi\_page*() and *PDF\_info\_pdi\_page*(): *adjustpage, gstate*
- ▶ Option for abbreviated structure element tagging according to Table 14.5: *tag*
- ▶ Option for selecting one of the page's top-level structure elements to retrieve some information from it: *index*

**Returns** The value of some page property as requested by *keyword*. If the requested property is not available for the page, the function returns 0. If an object handle is requested (e.g. *clippingpath*) this function will return a handle to the object, or -1 (in PHP: 0) if the object is not available. If the requested keyword produces *text*, a string index is returned, and the corresponding string must be retrieved with *PDF\_get\_string*().

**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.

**PDF/UA** The check for *fittingpossible* is stricter than in non-PDF/UA mode.

**Scope** any except *object*

Table 10.4 Keywords for `PDF_info_pdi_page()`

keyword	explanation
<b>fittingpossible</b>	<p>(Only relevant for Tagged PDF output) 0 if the page cannot be placed (i.e. <code>PDF_fit_pdi_page()</code> would throw an exception) for one of the following reasons:</p> <ul style="list-style-type: none"> <li>▶ One of the page's top-level tags is not allowed under the currently active tag according to the nesting rules for structure elements.</li> <li>▶ The page is untagged or contains no structure elements, and direct content is not allowed as child of the currently active tag.</li> <li>▶ The page has already been placed.</li> <li>▶ PDF/UA with weak document structure: there is a gap in the heading level numbers between the current structure element and its parents, and the imported structure sub-hierarchy.</li> </ul> <p>The value 1 is returned if the page can be placed in the current context. The <code>tag</code> option of <code>PDF_fit_pdi_page()</code> can be supplied and is taken into account for the result. Only the <code>tagname</code> suboption of the <code>tag</code> option is evaluated; no other suboptions should be supplied.</p> <p>Since the result is valid only for the current context this keyword should be used immediately before attempting to place a page.</p>
<b>lang</b>	String index for the <code>Lang</code> attribute of one of the imported page's top-level structure elements, or -1 if no <code>Lang</code> attribute could be determined. The <code>index</code> option can be used to select one of the top-level elements if there is more than one.
<b>mirroringx, mirroringy</b>	Horizontal or vertical mirroring of the page (expressed as 1 or -1) according to the supplied options
<b>pageheight, pagewidth</b>	Original page height and 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>
<b>topleveltag</b>	String index for the name of one of the imported page's top-level structure elements if the page has been opened with <code>usetags=true</code> and contains marked content associated with a structure element, otherwise -1 (e.g., for a page representing an <code>Artifact</code> ). The <code>index</code> option can be used to select one of the top-level elements if there is more than one. If the tag is a custom element which is <code>rolemapped</code> in the imported document's <code>rolemap</code> , the corresponding standard element name is reported, and not the custom element name.
<b>topleveltagcount</b>	Number of structure elements at the top level of the imported page's structure hierarchy. The <code>lang</code> and <code>topleveltag</code> keywords can be used to retrieve information about these elements, using the <code>index</code> option to select one. 0 is returned if no structure elements are imported, either because the page is untagged or contained no marked content corresponding to a structure element.
<b>xid</b>	(Only for PDF/VT) String index for the <code>GTS_XID</code> entry of the page, or -1 if no <code>GTS_XID</code> value has been assigned. The <code>GTS_XID</code> string can be used in the <code>CIP4/Summary/Content/Referenced</code> metadata property for <code>DPM</code> .

Table 10.5 Option for `PDF_info_pdi_page()`

option	description
<b>index</b>	(Integer; only relevant for the <code>lang</code> and <code>topleveltag</code> keywords) Selects one of the page's top-level structure elements whose attribute is retrieved. The value must be in the range 0..(toplevelcount-1). Default: 0

## 10.3 Other PDI Processing

C++ Java C# `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: `o`).

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

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ PDI processing options according to Table 10.6: *action*, *block*

**Returns** The value 1 if the function succeeded, or an error code of -1 (in PHP: `o`) if the function call failed. If *errorpolicy=exception* this function will throw an exception in case of an error. If no Blocks were found on the input page for *action=copyallblocks* the function returns 1.

**PDF/A** The output intent can be set using this function with the *copyoutputintent* option or with `PDF_load_iccprofile()`. If only device-independent colors are used in the document no output intent is required.

**PDF/X** The output intent must be set using this function with the *copyoutputintent* option or with `PDF_load_iccprofile()`.

**Scope** *document* for *action=copyoutputintent*,  
*page* for *action=copyallblocks* and *action=copyblock*

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/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/5pg, but not the output. <b>copyallblocks</b> (Only available in PPS) Copy all PDFlib Blocks from a page of the input document to the current output page according to the <i>block</i> option. <b>copyblock</b> (Only available in PPS) Copy a PDFlib Block from a page of the input document to the current output page according to the <i>block</i> option.

Table 10.6 Options for `PDF_process_pdi()`

<b>key</b>	<b>explanation</b>
<b>block</b>	(Option list; required for <code>action=copyallblocks</code> and <code>action=copyblock</code> ) Specify details of the Block copying process. The following suboptions are supported: <b>blockname</b> (Name string; only for <code>action=copyblock</code> and required in this case) Name of the Block <b>outputblockname</b> (Name string; only for <code>action=copyblock</code> ) Name under which the Block will be stored in the output page. Default: value of the <code>blockname</code> option <b>pagenumber</b> (Integer; required) The 1-based number of the page in the input document on which the Block is located.

## 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 accepts 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 C# 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 C# 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*, *number*, *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 raises an exception if pCOS does not run in full mode and the type of the object is *string*. However, some objects can nevertheless be retrieved in restricted mode under certain conditions; see pCOS Path Reference for details.

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.

**Bindings** COM: the result will be provided as Unicode string in UTF-16 format. If no text is available an empty string will be returned.

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

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

---

C++ Java C# **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, or if the contents of encrypted attachments in an unencrypted document are queried and the attachment password has not been supplied.

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** COM: Most client programs will use the Variant type to hold the stream contents. JavaScript with COM does not allow to retrieve the length of the returned variant array (but it does work with other languages and COM).

C and C++ language bindings: 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-capable 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 true, the stream data will be compressed with the filter which is specified in the image's <i>filterinfo</i> pseudo object. If false, the stream data will be uncompressed. Default: true for all unsupported filters, false 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 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 interactively. Alternatively, Blocks can be created with PPS itself.

*Cookbook* A full code sample for filling Blocks with PPS 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()`, `PDF_fill_pdfblock()`, and `PDF_graphics_block()`. 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

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 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 hide 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 C# *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) 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). 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 Table 2.1)
- ▶ 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 123)
- ▶ Textline Blocks, i.e. the *textflow* property or option is *false*:  
all Textline options (see Section 5.1, »Single-Line Text with Textlines«, page 89)
- ▶ Textflow Blocks, i.e. the *textflow* property or option is *true*:  
all options for *PDF\_add/create\_textflow()* (see Section 5.2, »Multi-Line Text with Textflows«, page 95) and all options for *PDF\_fit\_textflow()* (see Table 5.12)
- ▶ Text Block options according to Table 11.2: *textflow, textflowhandle*
- ▶ Option for default contents: *defaulttext* (see PDFlib Tutorial)

**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), the Block has wrong type, or the Block requires a newer PPS 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* 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.

*PDF/UA* Block decoration, i.e. ruling and shading created according to the *backgroundcolor*, *bordercolor*, *linewidth* properties is automatically tagged as *Artifact*.

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

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 <i>Textline</i> and <i>Textflow</i> 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 <i>Textflow</i> Blocks, and false for <i>Textline</i> 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 <i>Textflow</i> 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 <i>Textflow</i> 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 <i>Text Preparation</i> , <i>Text Formatting</i> and <i>Appearance</i> property groups of the Block will be ignored if <i>textflow-handle</i> is supplied since the corresponding values used for creating the <i>Textflow</i> will be applied.

## 11.3 Image Blocks

---

C++ Java C# *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) 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 Table 2.1)
- ▶ 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 123)
- ▶ Options for image processing according to Table 9.3
- ▶ Option for default contents: *defaultimage* (see PDFlib Tutorial)

**Returns** -1 (in PHP: 0) if the named Block doesn't exist on the page, the Block cannot be filled, the Block has wrong type, or the Block requires a newer PPS 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* option.

**PDF/UA** All raster images must be tagged as *Artifact* or *Figure* with a preceding call to *PDF\_begin\_item()*.

Block decoration, i.e. ruling and shading created according to the *backgroundcolor, bordercolor, linewidth* properties is automatically tagged as *Artifact*.

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

## 11.4 PDF Blocks

---

C++ Java C# *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) 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 Table 2.1))
- ▶ 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 123)
- ▶ Options for page processing according to Table 9.3
- ▶ Options for default contents: *defaultpdf, defaultpdfpage* (see PDFlib Tutorial)

**Returns** -1 (in PHP: 0) if the named Block doesn't exist on the page, the Block cannot be filled, the Block has wrong type, or the Block requires a newer PPS 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* option.

**PDF/UA** Block decoration, i.e. ruling and shading created according to the *backgroundcolor, bordercolor, linewidth* properties is automatically tagged as *Artifact*.

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

## 11.5 Graphics Blocks

---

C++ Java C# *int fill\_graphicsblock(int page, String blockname, int contents, String optlist)*

Perl PHP *int fill\_graphicsblock(int page, string blockname, int contents, string optlist)*

C *int PDF\_fill\_graphicsblock(PDF \*p, int page, const char \*blockname, int contents, const char \*optlist)*

---

Fill a graphics 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) Name of the Block.

**graphics** A valid graphics handle for the graphics to be filled into the Block, or -1 if the default graphics (as defined by Block properties) is to be used.

**optlist** An option list specifying graphics Block filling options. The following options are supported:

- ▶ General option: *errorpolicy* (see Table 2.1)
- ▶ 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 123)
- ▶ Options for graphics processing according to Table 9.3
- ▶ Option for default contents: *defaultgraphics* (see PDFlib Tutorial)

**Returns** -1 (in PHP: 0) if the named Block doesn't exist on the page, the Block cannot be filled, the Block has wrong type, or the Block requires a newer PPS 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 graphics referred to by the supplied graphics handle will be placed in the Block, subject to the Block's properties. If *graphics* is -1 (in PHP: 0) the function will use the Block's default graphics 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* option.

**PDF/UA** Block decoration, i.e. ruling and shading created according to the *backgroundcolor, bordercolor, linewidth* properties is automatically tagged as *Artifact*.

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



# 12 Interactive Features

## 12.1 Bookmarks

---

++ Java C# *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) Text for the bookmark.

**len** (C language binding only) Length of *text* (in bytes). 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 2.1), *hypertextencoding* and *hypertextformat* (see Table 2.3)
- ▶ Bookmark control options according to Table 12.1:  
*action*, *destination*, *destname*, *fontstyle*, *index*, *item*, *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 is specified the bookmark points to the current page (or the least recently generated 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.

**PDF/UA** Creating bookmarks is recommended for PDF/UA.

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

Table 12.1 Options for *PDF\_create\_bookmark()*

<b>option</b>	<b>explanation</b>
<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 12.10. 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> . <i>Destination</i> 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. This may be useful for separator bookmark.
<b>fontstyle</b>	(Keyword) Specifies the font style of the bookmark text: normal, bold, italic, bolditalic. Default: normal

Table 12.1 Options for `PDF_create_bookmark()`

<b>option</b>	<b>explanation</b>
<b>index</b>	(Integer) Index where to insert the bookmark within the parent. Values between 0 and the number of bookmarks of the same level will be used to insert the bookmark at that specific location within the parent. The value -1 can be used to insert the bookmark as the last one on the current level. Default: -1. However, for inserted or resumed pages bookmarks will be placed as if all pages had been generated in their physical order (i.e. the bookmarks reflect the page order).
<b>item</b>	(Item handle or keyword; the handle must not refer to an active structure element, but not to an inline or pseudo element; only for Tagged PDF) Handle for a structure item which will be associated with the bookmark. The value 0 always refers to the structure tree root. The value -1 and the equivalent keyword <code>current</code> refer to the currently active element.
<b>open</b>	(Boolean) If false, subordinate bookmarks will not be visible. If true, all children will be folded out. Default: false
<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: 0
<b>textcolor</b>	(Color) Specifies the color of the bookmark text. Supported color spaces: none, gray, rgb. Default: <code>rgb {0 0 0}</code> (=black)

## 12.2 Annotations

---

```
C++ Java C# 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* 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 12.2. Markup annotations are marked in the table since certain options apply only to markup annotations.

Table 12.2 Annotation types

type	notes; options specific for this type (in addition to general options)
<b>3D</b>	(PDF 1.6) 3D model: 3Dactivate, 3Ddata, 3Dinteractive, 3Dshared, 3Dinitialview
<b>Circle<sup>1</sup></b>	cloudy, createrichtext, inreplyto, interiorcolor, replyto
<b>File-Attachment<sup>1</sup></b>	attachment, calloutline, createrichtext, iconname, inreplyto, 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) attachment, movieposter, playmode, showcontrols, soundvolume, window-position, window-scale
<b>Polygon<sup>1</sup></b>	(PDF 1.5; vertices connected by straight lines): cloudy, createrichtext, inreplyto, polylinelist, replyto
<b>PolyLine<sup>1</sup></b>	(PDF 1.5; similar to polygons, except that the first and last vertices are not connected) createrichtext, endingstyles, inreplyto, interiorcolor, polylinelist, replyto
<b>Popup</b>	open, parentname
<b>RichMedia</b>	(PDF 1.7ext3) richmedia
<b>Square<sup>1</sup></b>	cloudy, createrichtext, inreplyto, interiorcolor, replyto
<b>Squiggly<sup>1</sup></b>	(squiggly-underline annotation) createrichtext, inreplyto, polylinelist, replyto

Table 12.2 Annotation types

type	notes; options specific for this type (in addition to general options)
<b>Stamp</b> <sup>1</sup>	createrichtext, iconname, inreplyto, orientate, replyto
<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
<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:

- ▶ General option: *hypertextencoding* (see Table 2.3)
- ▶ The following common options according to Table 12.3 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 12.3 are supported only for some annotation types according to Table 12.2:  
*alignment, calloutline, captionoffset, captionposition, createrichtext, destname, endingstyles, fillcolor, font, fontsize, highlight, iconname, inreplyto, interiorcolor, leaderlength, leaderoffset, line, movieposter, open, orientate, parentname, playmode, polylinelist, replyto, showcaption, showcontrols, soundvolume, windowposition, windowyscale*
- ▶ Options for *type=3D* according to Table 13.4:  
*3Dactivate, 3Ddata, 3Dinteractive, 3Dshared, 3Dinitialview*
- ▶ The *richmedia* option for *type=RichMedia* according to Table 13.4 with related suboptions in Table 13.7.
- ▶ Option for abbreviated structure element tagging according to Table 14.5: *tag*

**Details** In tagged PDF mode this function automatically creates a suitable *OBJR* element for the generated annotation. The user must create a corresponding *Link* or *Annot* container element (see PDFlib Tutorial) before calling this function.

**PDF/A** PDF/A-1: only the following annotation types are allowed:  
*Circle, FreeText, Highlight, Ink, Line, Link, Popup, Square, Squiggly, Stamp, StrikeOut, Text, Underline*

PDF/A-2/3: only *type=Link* is allowed.

Some options are restricted, see Table 12.3.

**PDF/X** Annotations are only allowed if they are positioned completely outside of the *BleedBox* (or *TrimBox/ArtBox* if no *BleedBox* is present).

PDF/X-1a/3: the annotation type *FileAttachment* is not allowed.

**PDF/UA** A non-grouping structure element must be created with *PDF\_begin\_item()* or the *tag* option when this function is called to create a visible annotation.

The option *contents* or the option *tag* with the suboption *ActualText* must be supplied for visible annotations.

**Scope** *page*

Table 12.3 Options for `PDF_create_annotation()`

option	explanation
<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. Default: white for type=Square, Circle, otherwise none PDF/A-1: this option can only be used if an RGB output intent has been specified, and gray or rgb color must be used.</p>
<b>attachment</b>	<p>(Asset handle; only for type=FileAttachment and Movie; required) File attachment which has been loaded with <code>PDF_load_asset()</code> and type=attachment. Supported suboptions: see Table 13.6.</p> <p>For type=FileAttachment: the file associated with the annotation</p> <p>For type=Movie: the media file in one of the following formats: AVI or QuickTime movie, WAV or AIFF sound. Note that the attachment must have been loaded with type=Attachment and external=true in <code>PDF_load_asset()</code>.</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>

Table 12.3 Options for `PDF_create_annotation()`

<b>option</b>	<b>explanation</b>
<b>contents</b>	(String for type=FreeText, otherwise Hypertext string with a maximum length of 65535 bytes; PDF/UA: required if no ActualText is supplied in the tag option, and always required for type=Link) 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. PDF/A-1a/2a/3a: recommended for annotations which do not display text.
<b>createdate</b>	(Boolean; PDF 1.5) If true, a date/time entry will be created for the annotation. Default: true for Markup annotations, false otherwise
<b>createrich-text</b>	(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><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 <code>PDF_fit_textflow/table()</code> before the call to <code>PDF_create_annotation()</code> 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 <code>PDF_fit_textflow/table()</code>.</p> <p><b>userunit</b> (Keyword) Measurement unit for scalar values (e.g. <code>firstlinedist</code>, <code>fontsize</code>): 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: <code>nextline</code>, <code>alignment</code>, <code>fillcolor</code>, <code>underline</code>, <code>strikeout</code>, <code>font</code>, <code>fontsize</code>, <code>textrise</code>, <code>text formatting options</code></p> <p>Note: setting font and alignment doesn't have the expected effect in Acrobat.</p>
<b>custom</b>	(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. Supported suboptions: <p><b>key</b> (String; required) Name of the dictionary key (excluding the <code>/</code> character). Any non-standard PDF key can be specified, as well as the following standard keys: <code>Contents</code>, <code>Name</code> (option <code>iconname</code>), <code>NM</code> (option <code>name</code>), and <code>Open</code>. 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 <code>boolean</code>, <code>name</code>, or <code>string</code></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>	(List of two non-negative floats; only for <code>borderstyle=dashed</code> ). The lengths of dashes and gaps for a dashed border in default units (see Table 7.1). Default: 3 3
<b>destination</b>	(Option list; only for type=Link; will be ignored if an activate action has been specified) Option list according to Table 12.10 defining the destination to jump to
<b>destname</b>	(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 <code>PDF_add_nameddest()</code> . Actions created with the <code>destination</code> or <code>destname</code> options of <code>PDF_create_action()</code> are dominant over this option.
<b>display</b>	(Keyword; will be forced to visible in PDF/A) Visibility on screen and paper: <code>visible</code> , <code>hidden</code> , <code>noview</code> , <code>noprint</code> . Default: <code>visible</code>
<b>endingstyles</b>	(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}): <p><code>none</code>, <code>square</code>, <code>circle</code>, <code>diamond</code>, <code>openarrow</code>, <code>closedarrow</code></p> <p>Additionally for PDF 1.5: <code>butt</code>, <code>openarrow</code>, <code>rclosedarrow</code></p> <p>Additionally for PDF 1.6: <code>slash</code></p>
<b>filename</b>	Deprecated, use <code>attachment</code>

Table 12.3 Options for `PDF_create_annotation()`












option	explanation
<b>fillcolor</b>	(Color; only for type=FreeText) Fill color of the text. Supported color spaces are none, gray, rgb, and (in PDF 1.6) cmyk. Default: {gray 0} (=black) PDF/A-1: this option can only be used if an RGB output intent has been specified, and gray or rgb color must be used.
<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. The value 0 or keyword auto means that Acrobat adjusts 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>	(String; only for type=Text, Stamp, FileAttachment) Name of an icon to be used in displaying the annotation (to create an annotation without any visible icon set opacity=0): For type=Text (default: note): comment  , key  , note  , help  , newparagraph  , paragraph  , insert  For type=Stamp, but note that these don't work reliably in Adobe Reader; the template option is recommended instead (default: draft): approved, experimental, notapproved, asis, expired, notforpublicrelease, confidential, final, sold, departmental, forcomment, topsecret, draft, forpublicrelease For type=FileAttachment (default: pushpin): graph  , pushpin  , paperclip  , tag  The template option can be used to create custom icons.
<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. Default: none PDF/A-1: this option can only be used if an RGB output intent has been specified, and gray or rgb color must be used.
<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>	(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). Leader lines are auxiliary lines which are drawn from each endpoint of the line perpendicular to the line itself. The length is specified by two numbers (default: {0 0}): 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. The second value, which can be omitted, represents the length of leader line extension which is drawn on the opposite side of the line. A positive value is ignored if the first value is 0.
<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

Table 12.3 Options for `PDF_create_annotation()`

<b>option</b>	<b>explanation</b>
<b>line</b>	(Line; only for type=Line; required) A list of four numbers x1, y1, x2, y2 specifying the start and end points of the line in default coordinates (if the usercoordinates option 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
<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>	Deprecated, use attachment
<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; not for PDF/A-1) 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 popup annotation. If this option is supplied, the options contents, annotcolor and title of the parent annotation override the corresponding values of the popup 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



Table 12.3 Options for `PDF_create_annotation()`

option	explanation
<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 <code>inreplyto</code> option. Supported keywords (default: <code>reply</code> ): <b>reply</b> The annotation must be considered a reply to the annotation specified by <code>inreplyto</code> . <b>group</b> The annotation must be grouped with the annotation specified by <code>inreplyto</code> .
<b>richmedia</b>	(Option list; required for <code>type=RichMedia</code> ) Rich media options according to Table 13.7
<b>rotate</b>	(Boolean; must not be set to true for text annotations in PDF/A) 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, otherwise true
<b>showcaption</b>	(Boolean; only for <code>type=Line</code> ; PDF 1.6) If true, the text specified in the <code>contents</code> or <code>createrichtext</code> options will be replicated as a caption in the appearance of the line. Default: false
<b>showcontrols</b>	(Boolean; only for <code>type=Movie</code> ) If true a controller bar while playing the movie or sound will be displayed. Default: false
<b>soundvolume</b>	(Float; only for <code>type=Movie</code> ) 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>	(Option list) Visual appearance of the annotation for one or more states. This may be useful e.g. for custom stamps. Supported suboptions: <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 <b>fitmethod</b> (Keyword) Method to fit the template into the annotation rectangle. If <code>fitmethod</code> is different from <code>entire</code> the annotation rectangle will be adapted to the template box (default: <code>entire</code> ): <b>nofit</b> Position the template only, without any scaling or clipping. <b>meet</b> Position the template according to the <code>position</code> 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. <b>entire</b> Position the template according to the <code>position</code> option, and scale it such that it entirely covers the rectangle. Generally this method will distort the template. <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 <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. Default: {left bottom}.
<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 <code>title</code> is 255 single-byte characters or 126 Unicode characters. However, a practical limit of 32 characters is advised. Default: none

Table 12.3 Options for `PDF_create_annotation()`

<b>option</b>	<b>explanation</b>
<b>usematchbox</b>	<p>(List of strings) The <code>llx/llly/urx/ury</code> parameters are ignored and the named matchbox is 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 <code>all</code> to specify all rectangles referring to the selected matchbox. If the second element is missing, it defaults to <code>all</code>.</p> <p>If the matchbox itself or the specified rectangle does not exist on the current page, the function silently returns without creating any annotation. When using matchboxes to create annotations in a table cell <code>PDF_create_annotation()</code> must be called after <code>PDF_fit_table()</code> to make sure that the matchbox size has already been calculated.</p>
<b>user-coordinates</b>	<p>(Boolean) If <code>false</code>, 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 <code>usercoordinates</code> option</p>
<b>window-position</b>	<p>(List of 2 floats or keywords; only for <code>type=Movie</code>) 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 <code>{0 0}</code> denoting the lower left corner of the screen and <code>{100 100}</code> the upper right corner. The keywords <code>left</code>, <code>center</code>, <code>right</code> (in horizontal screen direction) or <code>bottom</code>, <code>center</code>, <code>top</code> (in vertical screen direction) can be used as equivalents for the values <code>0</code>, <code>50</code>, and <code>100</code>. Default: <code>{center center}</code></p>
<b>window-scale</b>	<p>(Float or keyword; only for <code>type=Movie</code>) 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):</p> <p><b>fullscreen</b> The movie will be played using all of the available screen area.</p>
<b>zoom</b>	<p>(Boolean; must not be set to <code>true</code> for text annotations in PDF/A) If <code>true</code>, 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: <code>false</code> for text annotations in PDF/A, otherwise <code>true</code></p>

## 12.3 Form Fields

*Cookbook* A full code sample can be found in the *Cookbook topic* `webserver/starter_webform`.

---

C++ Java C# **`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` 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 coordinates 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). If `len = 0` a null-terminated string must be provided.

**`type`** The field type according to Table 12.4.

Table 12.4 Form field types

<b><code>type</code></b>	<b><code>icon</code></b>	<b>Options specific for this type (in addition to general options)</b>
<b><code>pushbutton</code></b>		<code>buttonlayout</code> , <code>caption</code> , <code>captiondown</code> , <code>captionrollover</code> , <code>charspacing</code> , <code>fitmethod</code> , <code>icon</code> , <code>icondown</code> , <code>iconrollover</code> , <code>position</code> , <code>submitname</code>
<b><code>checkbox</code></b>		<code>currentvalue</code> , <code>itemname</code>
<b><code>radiobutton</code></b>		<code>buttonstyle</code> , <code>currentvalue</code> , <code>itemname</code> , <code>toggle</code> , <code>unisonselect</code> <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><code>listbox</code></b>		<code>charspacing</code> , <code>currentvalue</code> , <code>itemnamelist</code> , <code>itemtextlist</code> , <code>multiselect</code> , <code>sorted</code> , <code>topindex</code>
<b><code>combobox</code></b>		<code>commitonselect</code> , <code>charspacing</code> , <code>currentvalue</code> , <code>editable</code> , <code>itemnamelist</code> , <code>itemtextlist</code> , <code>sorted</code> , <code>spellcheck</code>
<b><code>textfield</code></b>		<code>comb</code> , <code>charspacing</code> , <code>currentvalue</code> , <code>fileselect</code> , <code>maxchar</code> , <code>multiline</code> , <code>password</code> , <code>richtext</code> , <code>scrollable</code> , <code>spellcheck</code>
		<i>Text fields are also used for barcodes:</i> <code>barcode</code>
<b><code>signature</code></b>		<code>charspacing</code> , <code>lockmode</code>

**optlist** An option list specifying field properties:

- ▶ General options: *errorpolicy* (see Table 2.1), *hypertextencoding* and *hypertextformat* (see Table 2.3)
- ▶ Options for field properties according to Table 12.5. The following options are supported for all field types:  
*action, alignment, backgroundcolor, barcode, bordercolor, borderstyle, calcolor, dasharray, defaultvalue, display, exportable, fieldtype, fillcolor, font, fontsize, highlight, layer, linewidth, locked, orientate, readonly, required, strokecolor, taborder, tooltip, usercoordinates*
- ▶ The options listed in Table 12.4 are supported for specific field types. They are also detailed in Table 12.5.
- ▶ (Not for *PDF\_create\_fieldgroup()*) Option for abbreviated structure element tagging according to Table 14.5: *tag*

**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 standard, 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 (see PDFlib Tutorial).

In tagged PDF mode this function automatically creates a suitable *OBJR* element for the generated form field. The user must create the corresponding *Form* container element (see PDFlib Tutorial) before calling this function.

**PDF/A** This function must not be called.

**PDF/X** Form fields are only allowed if they are positioned completely outside of the BleedBox (or TrimBox/ArtBox if no BleedBox is present).

**PDF/UA** A structure element of type *Form* must be created with *PDF\_begin\_item()* or the *tag* option when this function is called. The *tooltip* option is required.

**Scope** *page*

---

C++ Java C# **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). 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 becomes part of this group. All field property options provided in the `optlist` for a group are inherited by all fields belonging to this group.

**PDF/A** This function must not be called.

**PDF/UA** The `tooltip` option is required. The `ZapfDingbats` font is required for `type=radiobutton` and `checkbox`. Since all fonts must be embedded in PDF/UA, an embeddable font outline file for `ZapfDingbats` must be configured.

**Scope** any except `object`

Table 12.5 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>action</b>	(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 <code>type=pushbutton</code> , <code>checkbox</code> , <code>radiobutton</code> . Default: empty list
<b>activate</b>	Actions to be performed when the field is activated.
<b>blur</b>	Actions to be performed when the field loses the input focus.
<b>calculate</b>	JavaScript actions to be performed in order to recalculate the value of this field when the value of another field changes.
<b>close</b>	(PDF 1.5) Actions to be performed when the page containing the field is closed.
<b>down</b>	Actions to be performed when the mouse button is pressed inside the field's area.
<b>enter</b>	Actions to be performed when the mouse enters the field's area.
<b>exit</b>	Actions to be performed when the mouse exits the field's area.
<b>focus</b>	Actions to be performed when the field receives the input focus.
<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.
<b>invisible</b>	(PDF 1.5) Actions to be performed when the page containing the field is no longer visible.
<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.
<b>open</b>	(PDF 1.5) Actions to be performed when the page containing the field is opened.
<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).
<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.
<b>visible</b>	(PDF 1.5) Actions to be performed when the page containing the field becomes visible.
<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>	

Table 12.5 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

<b>option</b>	<b>explanation</b>
<b>barcode</b>	<p>(Option list; only for <code>type=textfield</code>; implies <code>readonly</code>; PDF 1.7ext3) Create a barcode field according to the options in Table 12.6. 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:</p> <pre>action={calculate=...}</pre> <p>The barcode will be rendered in Acrobat 9 or above, but not 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 <code>solid</code> , <code>beveled</code> , <code>dashed</code> , <code>inset</code> , <code>underline</code> . Default: <code>solid</code>
<b>button-layout</b>	(Keyword; only for <code>type=pushbutton</code> ) The position of the button caption relative to the button icon, provided both have been specified: <code>below</code> , <code>above</code> , <code>right</code> , <code>left</code> , <code>overlaid</code> . Default: <code>right</code>
<b>buttonstyle</b>	(Keyword; only for <code>type=radiobutton</code> and <code>checkbox</code> ) Specifies the symbol to be used for the field: <code>check</code> , <code>cross</code> , <code>diamond</code> , <code>circle</code> , <code>star</code> , <code>square</code> . Default: <code>check</code>
<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 <code>calcorder</code> used on the current page (and 10 initially)
<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 is visible when the button doesn't have input focus. It is displayed with the font supplied in the <code>font</code> option. An empty string (i.e. <code>caption {}</code> ) produces neither caption nor icon. Default: <code>none</code>
<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: <code>none</code>
<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: <code>none</code>
<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. Default: 0
<b>comb</b>	(Boolean; only for <code>type=textfield</code> ; PDF 1.5) If <code>true</code> and the <code>multiline</code> , <code>fileselect</code> , and <code>password</code> options are <code>false</code> , 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: <code>false</code>
<b>commit-onselect</b>	(Boolean; only for <code>type=listbox</code> , <code>combobox</code> ; PDF 1.5) If <code>true</code> , an item selected in the list will be committed immediately upon selection. If <code>false</code> , the item will only be committed upon exiting the field. Default: <code>false</code>
<b>currentvalue</b>	<p>(Not for <code>type=pushbutton</code>, <code>signature</code>) The field's initial value. Type and default depend on the field type:</p> <p><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></p> <p><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: <code>empty</code></p> <p><b>listbox</b> (List of integers) Indices of the selected items within <code>itemtextlist</code>. Default: <code>none</code></p>
<b>dasharray</b>	(List of two non-negative floats; only for <code>borderstyle=dashed</code> ). The lengths of dashes and gaps for a dashed border in default units (see Table 7.1). Default: 3 3

Table 12.5 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>defaultvalue</b>	The field's value after a <code>ResetForm</code> 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 type <code>combobox</code> ) If <code>true</code> , the currently selected text in the box can be edited. Default: <code>false</code>
<b>exportable</b>	(Boolean) The field will be exported when a <code>SubmitForm</code> action happens. Default: <code>true</code>
<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 <code>true</code> , the text in the field will be treated as a file name. Default: <code>false</code>
<b>fillcolor</b>	(Color) Fill color for text. Supported color spaces: <code>gray</code> , <code>rgb</code> , <code>cmymk</code> . Default: <code>{gray 0}</code> (=black)
<b>fitmethod</b>	(Keyword; only for <code>type=pushbutton</code> ) Method of placing a template provided with the <code>icon</code> , <code>icondown</code> , and <code>iconrollover</code> options within the button. Supported keywords (default: <code>meet</code> ): <b>auto</b> same as <code>meet</code> if the template fits into the button, otherwise <code>clip</code> <b>nofit</b> same as <code>clip</code> <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 <code>meet</code> <b>entire</b> template will be scaled so that it fully fits into the button
<b>font</b>	(Font handle). Font to be used for the field. Acrobat can display characters even if they are not included in the font's encoding. For example, you can use <code>encoding=winansi</code> and supply Unicode characters outside <code>winansi</code> . Font usage depends on the field type: <ul style="list-style-type: none"> <li>▶ Fields with <code>type=listbox</code>, <code>combobox</code>, or <code>textfield</code>: this option is required.</li> <li>▶ For <code>type=pushbutton</code> this option is required if one or more of the <code>caption</code>, <code>captionrollover</code>, or <code>captiondown</code> options are specified.</li> <li>▶ Fields with <code>type=radiobutton</code> or <code>checkbox</code> always use <code>ZapfDingbats</code>.</li> </ul>
<b>fontsize</b>	(FontSize) Font size in user coordinates. The value <code>0</code> or keyword <code>auto</code> which means that Acrobat will adjust the <code>fontsize</code> to the rectangle. Default: <code>auto</code>
<b>highlight</b>	(Keyword) Highlight mode of the field when the user clicks on it: <code>none</code> , <code>invert</code> , <code>outline</code> , <code>push</code> . Default: <code>invert</code>
<b>icon</b>	(Template handle <sup>1</sup> ; only for <code>type=pushbutton</code> ; one of the <code>caption</code> or <code>icon</code> options must be supplied for push buttons) Handle for a template which will be visible when the button doesn't have input focus. Default: <code>none</code>
<b>icondown</b>	(Template handle <sup>1</sup> ; only for <code>type=pushbutton</code> ) Handle for a template which will be visible when the button is activated. Default: <code>none</code>
<b>iconrollover</b>	(Template handle <sup>1</sup> ; only for <code>type=pushbutton</code> ) Handle for a template which will be visible when the button has input focus. Default: <code>none</code>
<b>itemname</b>	(Hypertext string; only for <code>type=radiobutton</code> , <code>checkbox</code> ; 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: <code>field name</code>

Table 12.5 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

<b>option</b>	<b>explanation</b>
<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 coordinates. 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
<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).



Table 12.5 Options for field properties with `PDF_create_field()` and `PDF_create_fieldgroup()`

option	explanation
<b>submitname</b>	(Hypertext string; recommended only for <code>type=pushbutton</code> ) 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 <code>taborder</code> 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 <code>PDF_create_fieldgroup()</code> and <code>type=radiobutton</code> ) 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; non-empty value required in PDF/UA) The text visible in the field's tooltip, also used by screen readers. For radio buttons and groups Acrobat uses the tooltip of the first button in the group, others are ignored. Default: none
<b>topindex</b>	(Integer; only for <code>type=listbox</code> ) Index of the first visible entry. The first item has index 0. Default: 0
<b>unisonselect</b>	(Boolean; only for <code>PDF_create_fieldgroup()</code> , <code>type=radiobutton</code> and PDF 1.5) If true, radio buttons with the same field name or item name will be selected simultaneously. Default: false
<b>user-coordinates</b>	(Boolean) If false, field coordinates are expected in the default coordinate system; otherwise the current user coordinate system will be used. Default: the value of the <code>usercoordinates</code> global option

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 12.6 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>symbolology=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>symbolology=PDF417</code> the values must be in the range 0-8; for <code>symbolology=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>symbolology</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>symbolology=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

## 12.4 Actions

C++ Java C# *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 12.7.

Table 12.7 Action types

<b>type</b>	<b>notes; options allowed for this type</b>
<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: destination, destname, filename, newwindow, targetpath
<b>GoToR</b>	Go to a destination in another (remote) document: destination, destname, filename, newwindow
<b>Hide</b>	Hide or show an annotation or form field: hide, namelist
<b>ImportData</b>	Import form field values from a file: filename
<b>JavaScript</b>	Execute a script with JavaScript code: script, scriptname
<b>Launch</b>	Launch an application or document: defaultdir, filename, newwindow, operation, parameters
<b>Movie</b>	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>	Set some or all form fields to their default values: exclude
<b>RichMedia-Execute</b>	(PDF 1.7ext3) Send a command to a RichMedia annotation: functionname, instance, richmediaargs, target
<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 2.1) and *hypertextencoding* (see Table 2.3)
- ▶ The following type-specific options according to Table 12.8 are supported for specific action types according to Table 12.7:

*3Dview, richmediaargs, canonicaldate, defaultdir, destination, destname, duration, exclude, exportmethod, filename, functionname, hide, instance, ismap, layerstate, menuname, namelist, newwindow, operation, parameters, preserveradio, script, scriptname, submit-emptyfields, target, targetpath, transition, url*

**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, book-marks) 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/A** Only the following action types are allowed:  
*GoTo, GoToE, GoToR, Named, RichMediaExecute, SubmitForm, URI*

**PDF/X** This function must not be called.

**PDF/UA** The *ismap=true* option is not allowed.

**Scope** any except *object*. The returned handle can be used until the next call to *PDF\_end\_document()*.

Table 12.8 Options for action properties with *PDF.create\_action()*

<b>option</b>	<b>explanation</b>
<b>3Dview</b>	(Keyword or 3D view handle; GoTo3DView; required) Selects the view of a 3D annotation; One of the keywords <i>first, last, next, previous</i> (referring to the respective entries in the annotation's <i>views</i> option), or <i>default</i> (referring to the annotation's <i>defaultview</i> 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 <i>destname</i> is supplied) Option list according to Table 12.10 defining the destination to jump to.
<b>destname</b>	(Hypertext string) GoTo (required unless <i>destination</i> is supplied): name of a destination which has been defined with <i>PDF.add_nameddest()</i> . The destination can be created before or after referring to it. GoToR, GoToE (required unless <i>destination</i> 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 <i>namelist</i> option specifies which fields to exclude; all fields in the document are submitted except those listed in the <i>namelist</i> array and those whose <i>exportable</i> option is false. If false, the <i>namelist</i> 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 <i>namelist</i> option specifies which fields to exclude; all fields in the document are reset except those listed in the <i>namelist</i> array. If false, the <i>namelist</i> option specifies which fields to include in resetting. All members of specified field groups will be reset as well. Default: false

Table 12.8 Options for action properties with `PDF_create_action()`

<b>option</b>	<b>explanation</b>
<b>export-method</b>	<p>(Keyword list; SubmitForm) Controls how the field names and values are submitted. Default: <code>fdf</code>.</p> <p><b>html, fdf, xfdf, pdf</b> In HTML, FDF, XFDF, or PDF format, respectively</p> <p><b>annotfields</b> (Only for <code>fdf</code>) Include all annotations and fields.</p> <p><b>coordinate</b> (Only for <code>html</code>) The coordinates of the mouse click that caused the <code>submitform</code> 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.</p> <p><b>exclurl</b> (Only for <code>fdf</code>) The submitted FDF will exclude the url string.</p> <p><b>getrequest</b> (Only for <code>html</code> and <code>pdf</code>) Submit using HTTP GET; otherwise HTTP POST</p> <p><b>onlyuser</b> (Only for <code>fdf</code> and <code>annotfields</code>) The submit will include only those annotations whose name matches the name of the current user, as determined by the remote server.</p> <p><b>updates</b> (Only for <code>fdf</code>) Include all incremental updates contained in the underlying PDF document</p> <p>Example for combined options: <code>exportmethod {fdf updates onlyuser}</code></p>
<b>filename</b>	<p>(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 <code>\\server\volume</code>.</p> <p>ImportData (required): name of the external file containing forms data.</p> <p>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.</p>
<b>functionname</b>	<p>(Hypertext string; RichMediaExecute; required) String specifying the script command as a primitive ActionScript or JavaScript function name (not a full script). If the target instance specified by the <code>instance</code> option contains Flash content, the command string represents an ActionScript ExternalInterface call to the script engine context specific to the target instance. If the target instance is a 3D model, the call is made in the global context of the annotation's instance of the 3D JavaScript engine.</p>
<b>hide</b>	<p>(Boolean; Hide) Indicates whether to hide (true) or show (false) annotations. Default: <code>true</code></p>
<b>instance</b>	<p>(Integer; RichMediaExecute) Index of the option list (starting with 1) in the <code>instances</code> suboption of the configuration suboption of the <code>richmedia</code> option of <code>PDF_create_annotation()</code> to specify a Flash or 3D instance of the RichMedia annotation for which to execute the script. Default: <code>1</code></p>
<b>ismap</b>	<p>(Boolean; URI; true not allowed for PDF/UA) If <code>true</code>, the coordinates of the mouse position will be added to the target URI when the url is resolved. Default: <code>false</code></p>
<b>layerstate</b>	<p>(Option list; SetOCGState; required) List of pairs where each pair consists of a keyword and a layer handle. Supported keywords:</p> <p><b>on</b> Activate the layer</p> <p><b>off</b> Deactivate the layer</p> <p><b>toggle</b> Reverse the state of the layer. If this is used <code>preserveradio</code> should be set to <code>false</code>.</p>
<b>menuname</b>	<p>(String; Named; required) The name of the menu item to be performed. In PDF/A only <code>nextpage</code>, <code>prevpage</code>, <code>firstpage</code>, <code>lastpage</code> 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 <code>interactive/acrobat_menu_items</code>.</p>
<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 <code>false</code>.</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>

Table 12.8 Options for action properties with `PDF_create_action()`

option	explanation
<b>newwindow</b>	(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. Launch: This entry is ignored if the file is not a PDF document. Default: Acrobat behaves according to the current user preference.
<b>operation</b>	(Keyword; Launch) A keyword specifying the operation to be applied to the document specified in the filename option. This is only supported by Acrobat on Windows. If the filename option designates an application instead of a document, this option will be ignored and the application is launched. Supported keywords (default: open): <b>open</b> open a document <b>print</b> print a document  (Keyword; Movie) A keyword specifying the operation to be applied to the movie or sound. Supported keywords (default: play): <b>play</b> Start playing the movie, using the mode specified in the movie annotation's playmode option. If the movie is currently paused, it is repositioned to the beginning before playing. <b>stop</b> Stop playing the movie. <b>pause</b> Pause a playing movie. <b>resume</b> Resume a paused movie.
<b>parameters</b>	(String; Launch) A parameter string to be passed to the application specified with the filename option. This is only supported by Acrobat on Windows. Multiple parameters can be separated with a space character, but individual parameters must not contain any space characters. This option should be omitted if filename designates a document. Default: none
<b>preserve-radio</b>	(Boolean; SetOCGState) If true, preserve the radio-button state relationship between layers. Default: true
<b>richmediaargs</b>	(POCA container handle; RichMediaExecute) Handle for an array container which specifies an arbitrary number of arguments for the command. Valid arguments are objects of type string, integer, float, or Boolean. The array must have been created with the option usage=richmediaargs. Default: no arguments
<b>script</b>	(Hypertext string; JavaScript; required) A string containing the JavaScript code to be executed. In order to pass arbitrary strings with this option the option list syntax described in »Unquoted string values in option lists«, page 9, may be useful.
<b>scriptname</b>	(Hypertext string; JavaScript) If present, the JavaScript supplied in the script option will be inserted as a document-level JavaScript with the supplied name. If the same scriptname is supplied more than once in a document only the last script will be used. Document-level JavaScript will be executed after loading the document in Acrobat. This may be useful for scripts which are used in form fields.
<b>submit-emptyfields</b>	(Boolean; SubmitForm) If true, all fields characterized by the namelist and exclude options are submitted, regardless of whether they have a value. For fields without a value, only the field name is transmitted. If false, fields without a value are not submitted. Default: false
<b>target</b>	(String; GoTo3DView, Movie; RichMediaExecute; required) Name of the target 3D, movie, or rich media annotation for which to execute the script as specified in the name option of <code>PDF_create_annotation()</code> .
<b>targetpath</b>	(Option list; GoToE; required unless filename is specified) A target option list (see Table 12.9) 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.

Table 12.8 Options for action properties with `PDF_create_action()`

option	explanation
<b>transition</b>	(Keyword; Trans) Set the transition effect; see Table 3.9 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 http://).

Table 12.9 Suboptions for the targetpath option of `PDF_create_action()`

option	explanation
<b>annotation</b>	(Hypertext string; required if relation=child 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 pagenumber or destname.
<b>destname</b>	(Hypertext string; required unless pagenumber is supplied and relation=child 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 pagenumber is specified.
<b>name</b>	(Hypertext string; required if relation=child and the target is located in the attachments list; otherwise it must be absent; will be ignored if annotation is specified) Name of the target in the attachments list of <code>PDF_begin/end_document()</code> .
<b>pagenumber</b>	(Integer; required unless destname is supplied and relation=child and the target is associated with a file attachment annotation; will be ignored if destname 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: <ul style="list-style-type: none"> <li><b>parent</b> The target is the parent of the current document.</li> <li><b>child</b> The target is a child of the current document.</li> </ul>
<b>targetpath</b>	(Option list) A target option list according to Table 12.9 specifying additional path information to the target document. If this option is absent the current document is the target file containing the destination.

## 12.5 Named Destinations

C++ Java C# **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 is used, the others are silently ignored.

**len** (C language binding only) Length of *name* (in bytes). 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 2.1), *hypertextencoding* and *hypertextformat* (see Table 2.3)
- ▶ Destination control options according to Table 12.10:  
*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** any except *object*

Table 12.10 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 page scope, and page 1 if in document scope. 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 12.10 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 ArtBox) 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.



## 12.6 PDF Packages and Portfolios

Portfolio features are implemented with the following functions and options:

- ▶ Portfolios can be created with the *portfolio* option of *PDF\_end\_document()*. This function is described in Section 3.1, »Document Functions«, page 41, the *portfolio* option is described below in Table 12.13.
- ▶ Files and folders can be added to a portfolio with *PDF\_add\_portfolio\_folder()* and *PDF\_add\_portfolio\_file()*. These functions are described below.
- ▶ Actions for navigating within a portfolio can be created with *PDF\_create\_action()* and *type=GoToE* (see Section 12.4, »Actions«, page 226).

---

C++ Java C# *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). 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 2.1), *hypertextencoding* and *hypertextformat* (see Table 2.3)
- ▶ Options for folder properties according to Table 13.6: *description*, *thumbnail*
- ▶ Metadata option according to Table 12.11: *fieldlist*

**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** any except *object*

Table 12.11 Options for *PDF\_add\_portfolio\_folder()* and *PDF\_add\_portfolio\_file()*

option	explanation
<i>fieldlist</i>	(List of option lists) Specify metadata fields for the file or folder. 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 12.12.

---

C++ Java C# `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 `filenamehandling` global option, see Table 2.3) 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). If `len=0` a null-terminated string must be provided.

**optlist** An option list specifying file properties:

- ▶ General options: `errorpolicy` (see Table 2.1) and `hypertextformat` (see Table 2.3)
- ▶ Options for file properties according to Table 13.6:  
`description`, `filename`, `mimetype`, `name`, `password`, `relationship`, `thumbnail`
- ▶ Metadata option according to Table 12.11: `fieldlist`

**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()`.

**PDF/A** PDF/A-1: this function must not be called.

PDF/A-2: `filename` must refer to a PDF/A-1 or PDF/A-2 document. Some options are restricted, see Table 13.6.

PDF/A-3: arbitrary file types can be added. The `relationship` option is required. Files added to a package are implicitly associated with the whole document.

**Scope** any except `object`

Table 12.12 Suboptions of the fieldlist option of `PDF_add_portfolio_folder()` and `PDF_add_portfolio_file()`

option	explanation
<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 <code>type</code> option and must match the corresponding <code>type</code> suboption of the schema suboption of the portfolio option.

Table 12.13 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 <code>coversheet</code> option. If a file with the <code>coversheet</code> name exists in multiple portfolio folders and no <code>coversheetfolder</code> has been specified, the first occurrence will be used. Default: none
<b>initialview</b>	(Keyword) Specifies the initial view. Supported keywords (default: detail): <b>custom</b> (PDF 1.7ext3; requires the <code>navigator</code> option) The portfolio is presented by a custom Flash-based navigator. <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«). <b>hidden</b> The portfolio is initially hidden, without preventing the user from obtaining a file list via explicit action (Acrobat: »Minimize view«). <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«).
<b>navigator</b>	(Option list; PDF 1.7ext3; required for <code>initialview=custom</code> ) Embed a custom Flash-based navigator in the portfolio. In order to actually use the custom navigator when the document is opened use <code>view=custom</code> ; otherwise the navigator can be used for editing the portfolio, but will not be active upon opening the document. Supported suboptions are listed in Table 12.14. The values of <code>category</code> , <code>description</code> , <code>icon</code> , and <code>name</code> will be used to present the navigator in the list of available portfolio layouts when the portfolio is edited in Acrobat.

Table 12.13 Suboptions of the portfolio option of `PDF_end_document()`

option	explanation
<b>schema</b>	<p>(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 <code>fieldlist</code> 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):</p> <p><b>editable</b> (Boolean) Specifies whether Acrobat should allow editing the field value. Default: false</p> <p><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>.</p> <p><b>label</b> (Hypertext string; required) The textual field label that is displayed to the user.</p> <p><b>order</b> (Integer) Relative order of the fields in the user interface (1,2,3,...)</p> <p><b>type</b> (Keyword) Data type of the field. The following types can be used to refer to user-defined fields in the <code>fieldlist</code> option (default: <code>text</code>):</p> <p><b>text</b> hypertext string</p> <p><b>date</b> PDF date string</p> <p><b>number</b> number</p> <p><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.</p>
<b>sort</b>	<p>(List of option lists, where each list contains a string and an optional keyword) Specifies the order in which the fields specified in the <code>schema</code> option will be sorted in the user interface. Each sublist contains the field name (required) and a keyword (optional). Supported keywords (Default: <code>ascending</code>):</p> <p><b>ascending</b> field values are sorted in ascending order</p> <p><b>descending</b> field values are sorted in descending order</p> <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 <code>schema</code> 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>
<b>split</b>	<p>(Option list; PDF 1.7ext3) Specifies the orientation and position of the splitter bar. The default depends on the <code>initialview</code> option: The value <code>detail</code> (or no value) implies horizontal orientation and <code>tile</code> indicates vertical orientation. No splitter is used if <code>initialview=hidden</code>. Supported suboptions:</p> <p><b>direction</b> (Keyword) Orientation of the splitter bar. Supported keywords:</p> <p><b>horizontal</b> Split the window horizontally.</p> <p><b>vertical</b> Split the window vertically.</p> <p><b>none</b> Don't split the window. The entire window is dedicated to the file navigation view.</p> <p><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 <code>direction=none</code>. Default: viewer dependent</p>

Table 12.14 Suboptions of the navigator suboption of the portfolio option of `PDF_add_portfolio_folder()` and `PDF_add_portfolio_file()`

option	explanation
<b>apiversion</b>	(String; required) Version of the navigator API required by the navigator SWF file, specified as a string of the form <code>m[.n[.p[.q]]]</code> , where <code>m</code> , <code>n</code> , <code>p</code> , and <code>q</code> are non-negative integers. If not present, <code>n</code> , <code>p</code> , and <code>q</code> default to 0. The following entries are recommended: for use with Acrobat 9: 9.0.0.0 for use with Acrobat X: 9.5.0.0
<b>assets</b>	(List of option lists, required) Assets that are used to implement the navigator, e.g. a Flash file, an icon and other resources such as images or XML files. Supported suboptions: <b>asset</b> (Asset handle; required) Handle for an asset loaded with <code>PDF_load_asset()</code> . <b>name</b> (Hypertext string with 1-255 characters; the characters : * " < >   must not be used; the last character must not be a period '); required) Name of the asset which can be used to identify it in Flash code.
<b>category</b>	(Hypertext string) Category in which the navigator will be presented
<b>description</b>	(Hypertext string) Description of the navigator
<b>flash</b>	(Hypertext string; required) Name of an asset in the assets option list with <code>type=Flash</code> , containing the SWF code which implements the portfolio layout.
<b>icon</b>	(Hypertext string) Name of an asset in the assets option list with <code>type=JPEG</code> or <code>PNG</code> , containing an icon for the navigator. However, the use of PNG images is recommended since JPEG images don't seem to work consistently in Acrobat. A size of 42x42 pixels is recommended for best results in Acrobat.
<b>id</b>	(String; required if <code>version</code> is specified) String representing a unique ID for the navigator, expressed as a URI. If a versioning scheme is to be implemented, it is necessary to provide the same <code>id</code> across all versions of the navigator. If no <code>id</code> is specified, PDFlib will generate a URN based on a machine-generated GUID.
<b>loadtype</b>	(Keyword) Method used to load the navigator SWF. Supported keywords (default is the keyword default): <b>module</b> The navigator SWF is loaded as Adobe Flex 2 module. <b>default</b> The navigator SWF is loaded as an ordinary SWF file.
<b>locale</b>	(String) String with a locale according to Unicode Technical Standard #35. Examples: <code>en_GB</code> , <code>de_DE</code> , <code>zh_Hans</code>
<b>name</b>	(Hypertext string; required) Name of the portfolio
<b>strings</b>	(List of pairs of hypertext strings) Localized strings for the navigator. Each pair consists of an identifier for the localized string and the localized string itself.
<b>version</b>	(String; requires the <code>id</code> option) Version of the navigator, specified as a string of the form <code>m[.n[.p[.q]]]</code> , where <code>m</code> , <code>n</code> , <code>p</code> , and <code>q</code> are non-negative integers. If not present, <code>n</code> , <code>p</code> , and <code>q</code> default to 0.

## 12.7 Geospatial Features

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 12.15 and subsequent tables specify the options for geospatial features in detail.

Table 12.15 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) Description of a world coordinate system associated with the viewport to use for geospatial measuring; see Table 12.16 for supported options.
<b>hypertext-encoding</b>	(Keyword) Specifies the encoding for the name option. An empty string is equivalent to unicode. Default: the value of the <i>hypertextencoding</i> global option
<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 12.16 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: deg): <b>degree</b> degrees <b>grad</b> grad (1/400 of the full circle, or 0.9 degrees)
<b>areaunit</b>	(Keyword) Specifies the preferred area display unit (default: sqm): <b>sqm</b> square meter <b>ha</b> hectar (10.000 square meters) <b>sqkm</b> square kilometer <b>sqft</b> square foot <b>a</b> acre <b>sqmi</b> square mile <i>The specified unit will be used for display only if the following Acrobat setting is disabled: »Preferences, Measuring (Geo), Use Default Area Unit«.</i>
<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 neatline). The points are expressed relative to the boundingbox of a page viewport or the extent of an image. Default: {0 0 0 1 1 1 0}, 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 12.17 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 <i>coords</i> option to specify the map.

Table 12.16 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 12.17.

Table 12.17 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.





# 13 Multimedia Features

## 13.1 3D Artwork

*Cookbook* A full code sample can be found in the *Cookbook* topic `multimedia/starter_3d`.

3D features are implemented with the following functions and options:

- ▶ 3D data can be loaded with `PDF_load_3ddata()`. This function is described below.
- ▶ 3D views can be created with `PDF_create_3dview()`. This function is described below.
- ▶ 3D annotations can be created with `PDF_create_annotation()` and `type=3D`. This function is described in Section 12.2, »Annotations«, page 211. However, the options of this function for controlling 3D annotations are described in Table 13.4 below.
- ▶ Actions for controlling 3D annotations can be created with `PDF_create_action()` and `type=3Dview` (see Section 12.4, »Actions«, page 226).

---

C++ Java C# `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 `filenamehandling` global option, see Table 2.3) Name of a disk-based or virtual file containing a 3D model.

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

**optlist** An option list specifying properties of the 3D model:

- ▶ General options: `errorpolicy` (see Table 2.1) and `hypertextencoding` (see Table 2.3)
- ▶ 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** The file must contain 3D data in PRC or U3D format.

**Scope** any except `object`. The returned handle can be used until the next call to `PDF_end_document()`.

Table 13.1 Options for `PDF_load_3ddata()`

option	explanation
<code>defaultview</code>	(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>
<code>script</code>	(Hypertext string) String containing JavaScript code to be executed when the 3D model is instantiated. Default: <code>no script</code>

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 <a href="http://www.ecma-international.org">www.ecma-international.org</a> ): 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 <code>PDF_create_3dview()</code> . The type option used when creating the views with <code>PDF_create_3dview()</code> must match the type option in <code>PDF_load_3ddata()</code> . Default: empty list

C++ Java C# `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). If *len* = 0 a null-terminated string must be provided.

**optlist** An option list specifying 3D view properties:

- ▶ General options: *errorpolicy* (see Table 2.1) and *hypertextencoding* (see Table 2.3)
- ▶ 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** any except *object*. 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 <code>camera2world</code> 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: <code>Artwork</code> ): <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 <code>GoTo</code> actions. This is an optional internal name which is treated separately from the required <code>username</code> 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 <code>PDF_load_3ddata()</code> with <code>type=PRC</code> ) Specify the type of 3D data (default: <code>U3D</code> ): <b>PRC</b> The view will be used in <code>PDF_load_3ddata()</code> with <code>type=PRC</code> . <b>U3D</b> The view will be used in <code>PDF_load_3ddata()</code> with <code>type=U3D</code> .
<b>U3Dpath</b>	(Hypertext string; will be ignored if the <code>camera2world</code> option is specified; only for <code>type=U3D</code> ) 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()`

<b>option</b>	<b>explanation</b>
<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 <code>backgroundcolor</code> refers to the current background color. Default: <code>backgroundcolor</code>
<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>	(Keyword; PDF 1.7) Render mode for displaying the 3D artwork (default: <code>Artwork</code> ): <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.

Table 13.4 3D options for `PDF_create_annotation()` with `type=3D`

option	explanation
<b>3Dactivate</b>	<p>(Option list; only for <code>type=3D</code>) Specifies when the 3D annotation should be activated and its state upon activation/deactivation. Supported suboptions:</p> <p><b>enable</b> (Keyword) Specifies when the animation should be enabled (default: <code>click</code>):</p> <ul style="list-style-type: none"> <li><b>open</b> Activate when the page is opened.</li> <li><b>visible</b> Activate when the page becomes visible.</li> <li><b>click</b> Annotation must explicitly be activated by a script or user action.</li> </ul> <p><b>enablestate</b> (Keyword) Initial animation state (default: <code>play</code>):</p> <ul style="list-style-type: none"> <li><b>pause</b> The 3D model is instantiated, but script animations are disabled.</li> <li><b>play</b> The 3D model is instantiated; script animations are enabled if present.</li> </ul> <p><b>disable</b> (Keyword) Specifies when the animation should be disabled (default: <code>invisible</code>):</p> <ul style="list-style-type: none"> <li><b>close</b> Deactivate when the page is closed.</li> <li><b>invisible</b> Deactivate when the page becomes invisible.</li> <li><b>click</b> Annotation must explicitly be deactivated by a script or user action.</li> </ul> <p><b>disablestate</b> (Keyword) State of the animation upon disabling (default: <code>reset</code>):</p> <ul style="list-style-type: none"> <li><b>pause</b> The 3D model can be rendered, but animations are disabled.</li> <li><b>play</b> The 3D model can be rendered and animations are enabled.</li> <li><b>reset</b> Initial state of the 3D model before it has been used in any way.</li> </ul> <p><b>modeltree</b> (Boolean; PDF 1.6) If <code>true</code>, the Model Tree navigation tab will be opened when the annotation is activated (default: <code>false</code>)</p> <p><b>toolbar</b> (Boolean; PDF 1.6) If <code>true</code>, the 3D toolbar (at the top of the annotation) will be displayed when the annotation is activated (default: <code>true</code>)</p>
<b>3Ddata</b>	(3D handle; only for <code>type=3D</code> ; required) 3D handle created with <code>PDF_load_3ddata()</code> .
<b>3Dinteractive</b>	(Boolean; only for <code>type=3D</code> ) If <code>true</code> , the 3D model is intended for interactive use. If <code>false</code> , it is intended to be manipulated with JavaScript. Default: <code>true</code>
<b>3Dshared</b>	(Boolean; only for <code>type=3D</code> ) If <code>true</code> , the 3D data specified in the <code>3Ddata</code> option will be referenced indirectly. Multiple 3D annotations which 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: <code>false</code>
<b>3Dinitialview</b>	(Keyword or 3D view handle) Specifies the initial view of the 3D model; One of the keywords <code>first</code> , <code>last</code> , (referring to the respective entries in the <code>views</code> option of <code>PDF_load_3ddata()</code> ), or <code>default</code> (referring to the model's <code>defaultview</code> option), or a 3D view handle created with <code>PDF_create_3dview()</code> . Default: <code>default</code>

## 13.2 Asset and Rich Media Features (Flash)

Multimedia features are implemented with the following functions and options:

- ▶ Multimedia assets for use in RichMedia annotations can be loaded with `PDF_load_asset()`. It can also be used to load assets which will be used as file attachments. This function is described below.
- ▶ RichMedia annotations can be created with `PDF_create_annotation()` and `type=RichMedia`. This function is described in Section 12.2, »Annotations«, page 211. However, the relevant suboptions of the `richmedia` option of this function are described in Table 13.7 and following tables below.
- ▶ Actions for controlling rich media annotations can be created with `PDF_create_action()` and `type=RichMediaExecute` (see Section 12.4, »Actions«, page 226).

Assets loaded with `PDF_load_asset()` can also be used with `PDF_create_annotation()` and `type=FileAttachment` or `Movie`.

---

C++ Java C# `int load_asset(String type, String filename, String optlist)`

Perl PHP `int load_asset(string type, string filename, string optlist)`

C `int PDF_load_asset(PDF *p, const char *type, const char *filename, int len, const char *optlist)`

---

Load a rich media asset or file attachment from a disk-based or virtual file.

**type** Keyword designating the type of the loaded asset according to Table 13.5.

Table 13.5 Asset types

type	allowed contents	asset can be used with function and option/suboption
3D	U3D and PRC	<code>PDF_create_annotation()</code> : richmedia/configurations/instances/asset
Attachment <sup>1</sup>	PDF/A-2: only PDF/A-1 and PDF/A-2; otherwise: any	<code>PDF_end_document()</code> : attachments <code>PDF_create_annotation()</code> : attachment PDF/A-3 only: <code>PDF_end_document()</code> , <code>PDF_begin/end_page_ext()</code> , <code>PDF_begin/end_dpart()</code> , <code>PDF_begin_template_ext()</code> , <code>PDF_load_image()</code> , <code>PDF_open_pdi_page()</code> , <code>PDF_load_graphics()</code> : associatedfiles
Flash	Shockwave (*.swf)	<code>PDF_create_annotation()</code> : richmedia/configurations/instances/asset <code>PDF_end_document()</code> : portfolio/navigator/flash
Generic	any	<code>PDF_create_annotation()</code> : richmedia/assets <code>PDF_end_document()</code> : portfolio/navigator/assets
JavaScript	text file containing ECMA Script edition 3 <sup>2</sup>	<code>PDF_create_annotation()</code> : richmedia/activate/script
JPEG	JPEG image	<code>PDF_end_document()</code> : portfolio/navigator/icon
PNG	PNG image	<code>PDF_end_document()</code> : portfolio/navigator/icon
Sound	MP3 etc.	<code>PDF_create_annotation()</code> : richmedia/configurations/instances/asset
Video	FLV, QuickTime, F4V, H.264 etc.	<code>PDF_create_annotation()</code> : richmedia/configurations/instances/asset

1. type=Attachment is implicitly assumed for `PDF_add_portfolio_folder/file()`, and for use as suboptions for the attachments option of `PDF_begin/end_document()`

2. in ISO 8859-1 encoding or UTF-16 LE or BE with BOM

**filename** (Name string; will be interpreted according to the *filenamehandling* global option, see Table 2.3) Name of a disk-based or virtual file which will be embedded in the PDF file. Unicode file names are supported, but require PDF 1.7 for correct display in Acrobat.

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

**optlist** An option list which may contain the following options:

- ▶ General option for all types: *errorpolicy* (see Table 2.1)
- ▶ If *type=Attachment* additional attachment property options according to Table 13.6 are supported:  
*description*, *documentattachment*, *external*, *filename*, *mimetype*, *name*, *password*, *relationshipthumbnail*

**Returns** An asset handle for rich media or file attachments which can be used with the functions listed in Table 13.5 until the end of the enclosing *document* scope. The returned asset handle cannot be reused across multiple PDF documents.

If *errorpolicy=return* the caller must check for a return value of -1 (in PHP: 0) since it signals an error. If the function call fails you can request the reason of the failure with *PDF\_get\_errmsg()*.

**Details** This function can be used with all PDF compatibility levels if *type=Attachment*, and requires PDF 1.7ext3 for all other types.

**PDF/A** PDF/A-1: this function must not be called.

PDF/A-2: *filename* must refer to a PDF/A-1 or PDF/A-2 document. Some options are restricted.

PDF/A-3: some options are restricted. Each attachment must be associated with exactly one part of the document, i.e. the generated asset handle must be supplied to exactly one *associatedfiles* option.

**PDF/X** PDF/X-1a/3: this function must not be called.

**Scope** any except *object*

Table 13.6 Options for *PDF\_load\_asset()* with *type=Attachment*, for *PDF\_add\_portfolio\_folder/file()*, and for use as suboptions for the attachments option of *PDF\_begin/end\_document()*

option	explanation
<b>description</b>	(Hypertext string; PDF 1.6; recommended for PDF/A-2/3 and PDF/UA) Descriptive text associated with the file.
<b>document-attachment</b>	(Boolean; only for PDF/A-3 and PDF 2.0 and only if the <i>relationship</i> option is specified; not for <i>PDF_add_portfolio_file/folder()</i> ) Save the associated file also as a document-level attachment (embedded file). This may be useful for listing the attachments in the user interface of PDF viewers which are not aware of associated files data structures (this includes Acrobat 9/X/XI). Default: false
<b>external</b>	(Boolean; must be false for PDF/A; not for the attachments option and <i>PDF_add_portfolio_file/folder()</i> ) If true, the contents of the file will not be embedded in the PDF, but only a reference to an external file will be created. The option <i>external=true</i> is required for movies to make sure that Acrobat will be able to play the movie. Default: false



Table 13.6 Options for `PDF_load_asset()` with `type=Attachment`, for `PDF_add_portfolio_folder/file()`, and for use as suboptions for the attachments option of `PDF_begin/end_document()`

option	explanation
<b>filename</b>	(Name string) Name of the file. The contents must conform to the requirements listed in Table 13.5 according to the specified type. Unicode file names are supported, but require PDF 1.7 for correct display in Acrobat. This option can alternatively be provided via the function parameter <code>filename</code> of <code>PDF_load_asset()</code> and <code>PDF_add_portfolio_file/folder()</code> , but is required if used with the attachments option of <code>PDF_begin/end_document()</code> . Only the base part of the filename without any directory components will be written to the PDF output.
<b>mimetype</b>	(String; required for PDF/A-3; not for <code>PDF_add_portfolio_folder()</code> ) MIME type of the file. If the MIME type is unknown, the string <code>application/octet-stream</code> must be used in PDF/A-3.
<b>name</b>	(Hypertext string; not for <code>PDF_add_portfolio_folder()</code> ) Name of the attachment. Default: <code>filename</code> without any path component
<b>password</b>	(String with up to 127 characters; not for PDF/A-2; only if PDI is available; not for <code>PDF_add_portfolio_folder()</code> ) PDF master password required to open a protected PDF document for fetching its date entries.
<b>relationship</b>	<p>(Hypertext string; only for PDF 2.0 and PDF/A-3; required for PDF/A-3; not for <code>PDF_add_portfolio_folder()</code>) Relationship of the file to the part of the document with which it is associated. The value may be an arbitrary string, but the following predefined keywords are listed in the PDF/A-3 standard:</p> <p><b>Alternative</b> The file is an alternative representation (e.g. audio).</p> <p><b>Data</b> The file represents information used to derive a visual presentation (e.g. CSV data for a table or graph).</p> <p><b>Source</b> The file is the original source material (e.g. word processor document associated with the document; spreadsheet associated with an image).</p> <p><b>Supplement</b> The file represents a supplemental representation of the original source or data which may be more easily consumable (e.g. MathML representation of an equation in an image).</p> <p><b>Unspecified</b> The relationship is not known or cannot be described with the other keywords.</p>
<b>thumbnail</b>	(Image handle) Image to be used as thumbnail for the file. 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> . Acrobat ignores the thumbnail for attachments.

Table 13.7 Suboptions for the richmedia option of `PDF_create_annotation()` with type=RichMedia

option	explanation
<b>activate</b>	(Option list) Option list according to Table 13.8 which specifies the style of presentation, default script behavior, default view information, and animation style when the annotation is activated.
<b>assets</b>	(List of option lists; required) Named asset which can be referenced from Flash content: <ul style="list-style-type: none"> <li><b>asset</b> (Asset handle; required) Handle for an asset loaded with <code>PDF_load_asset()</code>.</li> <li><b>name</b> (Hypertext string with 1-255 characters; the characters : * " &lt; &gt;   must not be used; the last character must not be a period !; required) Name of the asset which can be used to identify it in Flash code.</li> </ul>
<b>configuration</b>	(Option list, required) The configuration option list may contain one or more instance option lists. The type option helps to describe the behavior for the collection of those instances. For example, a FLV video file may require a SWF file to view it. Though the primary instance in the configuration may be a SWF file, the annotation is intended for video playback and thus should have a type of Video. Setting the type suggests the author's intended use of the assets, which better informs the choices when presenting content-specific user interfaces during the authoring or editing process. Supported options: <ul style="list-style-type: none"> <li><b>instances</b> (List of option lists; required) Each list specifies a single instance of an asset with settings to populate the artwork of an annotation. Supported options:               <ul style="list-style-type: none"> <li><b>asset</b> (Hypertext string; required) A rich media asset name specified in the assets option. Only names of rich media assets of type 3D, Flash, Sound, Video may be specified here.</li> <li><b>params</b> (Option list; only for type=Flash) Parameters related to a Flash asset according to Table 13.10</li> <li><b>name</b> (Hypertext string) Unique name of the configuration</li> <li><b>type</b> (Keyword) Primary content type for the configuration. Valid keywords are 3D, Flash, Sound, and Video. Default: the scene type is determined by referring to the type of the file specified in the first element of the instances option list.</li> </ul> </li> </ul>
<b>deactivate</b>	(Option list) Specifies the condition of unloading (restart or pause): <ul style="list-style-type: none"> <li><b>condition</b> (Keyword) Specifies when the annotation will be deactivated (default: clicked):               <ul style="list-style-type: none"> <li><b>clicked</b> The annotation is explicitly deactivated by a user action or script.</li> <li><b>closed</b> The annotation is deactivated as soon as the page that contains the annotation loses the focus as current page.</li> <li><b>invisible</b> The annotation is deactivated as soon as the page that contains the annotation is no longer visible.</li> </ul> </li> </ul>
<b>views</b>	(List of 3D view handles) 3D view handles returned by <code>PDF_create_3dview()</code> . If no views are specified, default values are used for the components of a 3D view, including rendering/lighting modes, background color, and camera data. Default: empty list

Table 13.8 Suboptions for the activate suboption of the richmedia option of `PDF_create_annotation()`

option	explanation
<b>animation</b>	(Option list) Preferred method to drive keyframe animations present in the artwork. Supported options: <ul style="list-style-type: none"> <li><b>playcount</b> (Integer) A nonnegative number represents the number of times the animation is played. A negative integer indicates that the animation is infinitely repeated. Default: -1</li> <li><b>speed</b> (Positive Float) A value greater than one shortens the time it takes to play the animation, or effectively speeds up the animation. This allows authors to change the desired speed of animations without re-authoring the content. Default: 1</li> <li><b>style</b> (Keyword) Specifies the animation style (default: none):               <ul style="list-style-type: none"> <li><b>none</b> Keyframe animations should not be driven directly by the viewer application. This value is used by documents that are intended to drive animations through alternate means such as JavaScript. The remaining suboptions of the animation option will be ignored.</li> <li><b>linear</b> Keyframe animations are driven linearly from begin to end. This results in a repetitive playthrough of the animation, such as in a walking motion.</li> <li><b>oscillating</b> Keyframe animations should oscillate along their time range. This results in a back-and-forth playing of the animation, such as exploding or collapsing parts.</li> </ul> </li> </ul>
<b>condition</b>	(Keyword) Specifies when the annotation will be activated (default: clicked): <ul style="list-style-type: none"> <li><b>clicked</b> The annotation is explicitly activated by a user action or script.</li> <li><b>opened</b> The annotation is activated as soon as the page that contains the annotation receives the focus as current page.</li> <li><b>visible</b> The annotation is activated as soon as any part of the page that contains the annotation becomes visible.</li> </ul>
<b>presentation</b>	(Option list) Specifies how the annotation and user interface elements will be laid out and drawn: Supported options are listed in Table 13.9.
<b>scripts</b>	(List of hypertext strings) List containing the names of rich media assets of type=JavaScript which have been embedded with <code>PDF_load_asset()</code> and specified in the assets suboption of the richmedia option. An empty list means that no script is executed. Default: empty list
<b>view</b>	(Keyword or 3D view handle) Specifies the activation view of 3D rich media. The handle must also be contained in the views suboption of the richmedia option list. If the views option was not specified default values for the components of a 3D view will be used. Supported keyword (default: first): <ul style="list-style-type: none"> <li><b>first</b> The first element in the views suboption of the richmedia option list</li> </ul>

Table 13.9 Suboptions for the presentation suboption of the activate suboption of the richmedia option of PDF\_create\_annotation()

option	explanation
<b>navigation-pane</b>	(Boolean) Default behavior of the navigation pane user interface element. If true the navigation pane is visible when the content is initially activated. Default: false
<b>passcontext-click</b>	(Boolean) Indicates whether a context click on the rich media annotation is passed to the media player runtime or is handled by the PDF viewer. If false the PDF viewer handles the context click. If true, the PDF viewer's context menu is not visible, and the user sees the context menu and any custom items generated by the media player runtime. Default: false
<b>style</b>	(Keyword) Specifies how the rich media will be presented (default: embedded): <b>embedded</b> embedded within the PDF page <b>windowed</b> in a separate window specified by the window option list
<b>toolbar</b>	(Boolean) Default behavior of an interactive toolbar associated with this annotation. If true a toolbar is displayed when the annotation is activated and given focus. Default: true for type=3D, false otherwise
<b>transparent</b>	(Boolean) Indicates whether the page content is displayed through the transparent areas of the rich media content. If true the rich media artwork is composited over the page content using an alpha channel. If false the rich media artwork is drawn over an opaque background prior to composition over the page content. Default: false
<b>window</b>	(Option list) Size and position of the floating window for style>windowed. Coordinates are expressed in user or default coordinates depending on the usercoordinates option. The behavior is similar to when the annotation options zoom and rotate are set to false. Supported suboptions: <b>heightdefault</b> (Float) Default height of the window. Default: 216 (in default coordinates) <b>widthdefault</b> (Float) Default width of the window. Default: 288 (in default coordinates)

Table 13.10 Suboptions for the params suboption of the instances suboption of the configurations suboption of the richmedia option of PDF\_create\_annotation()

option	explanation
<b>binding</b>	<p>(Keyword) Specifies to which entity the Flash content is bound (default: none):</p> <p><b>background</b> The Flash content is bound to the background and is rendered behind any 3D model content or Flash foreground elements in the active annotation. For a given RichMedia annotation there can be one active instance that has a params option list with binding=background. If more than one is specified, the last instance specified for the background is used.</p> <p><b>foreground</b> The Flash content is bound to the foreground and is rendered in front of the 3D model and the background Flash content in the active annotation. If more than one instance has binding=foreground, the Flash content is rendered in order from back to front, each instance composited over prior instances using its alpha channel.</p> <p><b>material</b> The Flash content is bound to a material that is part of 3D content. If that material is applied to geometry within a 3D scene, the Flash object appears to be playing upon this object as if conforming to the surface of the object.</p> <p><b>none</b> The Flash content is unbound and is not visible at playback time.</p>
<b>cuepoints</b>	<p>(List of option lists) A video can contain cue points that are encoded in the video stream or may be created by an associated ActionScript within the Flash content. Each option list specifies a state that relates the cue point to an action that may be passed to the application or may be used to change the appearance. Cue points in the Flash content are matched to the cue points declared in the PDF file by the values specified by the name or time options. Supported options:</p> <p><b>action</b> (Action list) Action that is executed if this cue point is triggered, meaning that the Flash content reached the matching cue point during its playback. Supported action trigger:  <b>activate</b> Actions to be performed if this cue point is triggered, meaning that the Flash content reached the matching cue point during its playback.</p> <p><b>name</b> (Text string; required) Name of the cue point to match against the cue point within the Flash content and for display purposes.</p> <p><b>time</b> (Float; required) The time value of the cue point in milliseconds to match against the cue point within the Flash content and for display purposes. Note that Acrobat 9 and X ignore the timestamp. However, since this behavior cannot be guaranteed for future versions of Acrobat a correct timestamp should be provided if known; otherwise a dummy value can be provided.</p> <p><b>type</b> (Keyword) Type of the cue point:  <b>Event</b> An event is a generic cue point of no specific significance other than that a corresponding action is triggered.  <b>Navigation</b>  A navigation cue point is an event encoded in a Flash movie (FLV). A chapter stop may be encoded so that when the user requests to go to or skip a chapter, a navigation cue point is used to indicate the location of the chapter.</p>
<b>flashvars</b>	<p>(Hypertext string) Formatted name value pairs passed to the Flash player context when activated. For the format specifics see the document »Using FlashVars to pass variables to a SWF«, TechNote tn_16417, available at <a href="http://www.adobe.com/go/tn_16417">www.adobe.com/go/tn_16417</a>. Default: no data is sent to the Flash player</p>
<b>materialname</b>	<p>(Hypertext string; required if binding=material) The material name that content is to be bound to.</p>
<b>settings</b>	<p>(Hypertext string) A text string used to store settings information associated with a Flash instance. This value is passed by the ActionScript ExternalInterface command multimedia_loadSettingsString. Default: undefined</p>



# 14 Document Interchange

## 14.1 Document Information Fields

---

C++ Java C# `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.

**len** (Only for `PDF_set_info2()`, and only for the C language binding) Length of *value* (in bytes). 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 fields supplied to `PDF_set_info()`.

Document info fields override corresponding properties in XMP document metadata supplied to the *metadata* option of `PDF_begin/end_document()`.

**PDF/X** Info fields with *key*=*Title* and *key*=*Creator* must be supplied with non-empty values. Alternatively in PDF/X-4 and PDF/X-5 the *metadata* option of `PDF_begin_document()` with the *dc:title* and *xmp:CreatorTool* XMP properties can be supplied.

Only the values *True* and *False* are allowed for the *Trapped* info field.

**PDF/UA** An info field with *key*=*Title* must be supplied with a non-empty value. Alternatively, the *metadata* option of `PDF_begin_document()` with the *dc:title* XMP property can be supplied.

**Scope** any; if used in *object* scope the supplied values will only be used for the next document.

Table 14.1 Keys for document information fields

<b>key</b>	<b>explanation</b>
<b>Subject</b>	Subject of the document
<b>Title</b>	Title of the document

<sup>1</sup> See [dublincore.org](http://dublincore.org)

Table 14.1 Keys for document information fields

<b>key</b>	<b>explanation</b>
<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«.
<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.
<b>any other name</b>	User-defined document information field. PDFlib supports an arbitrary number of fields. A custom field name should only be supplied once. See also moddate option of <i>PDF_begin/end_document()</i> . Custom document info fields must not contain any of the following characters if XMP metadata is created (via the <i>autoxmp</i> or <i>metadata</i> options): & \ < > " space Fields which are used for standard identification are not allowed.



## 14.2 XMP Metadata

XMP metadata can be supplied for the whole document or individual pages, fonts, ICC profiles, images, templates, and imported PDF pages. Table 14.2 lists suboptions for the *metadata* option of various functions.

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()` and the *templateoptions* option of `PDF_load_graphics()`

option	description
<b>compress</b>	(Boolean; not for <code>PDF_begin/end_document()</code> ) Compress the XMP metadata stream in the PDF output. If the option is only supplied in <code>PDF_begin_page_ext()</code> but not in <code>PDF_end_page_ext()</code> , its value takes precedence over the default. Default: false PDF/A-1 and PDF/X: <code>compress=true</code> is not allowed.
<b>inputencoding</b>	(Keyword) The encoding to interpret the metadata supplied in filename. Default: unicode
<b>inputformat</b>	(Keyword) The format of the metadata supplied in filename. Default: utf8, but bytes if <code>inputencoding</code> is an 8-bit encoding
<b>keepxmp</b>	(Boolean; only for <code>PDF_load_image()</code> and <code>PDF_load_graphics()</code> ; can not be combined with filename) XMP metadata present in an image or graphics file 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 as well as in SVG graphics. If no XMP metadata is found in the image or graphics file this option doesn't have any effect. Default: false
<b>filename</b>	(Name string; required unless <code>keepxmp</code> is supplied) The name of a file containing well-formed XMP metadata. It will be interpreted according to the <code>filenamehandling</code> global option, see Table 2.3.

**PDF/A** The XMP identification properties for PDF/A are created automatically.

PDFlib synchronizes relevant entries in user-supplied XMP streams to standard document info fields (similar to *autoxmp* mode which synchronizes document info fields to XMP). However, PDFlib does not synchronize other XMP entries to custom document info fields. Additional PDF/A requirements for XMP document metadata are discussed in the PDFlib Tutorial. The following validation will be applied to document metadata:

- ▶ PDF/A-1: XMP must conform to XMP 2004 or include an extension schema description;
- ▶ PDF/A-2/3: XMP must conform to XMP 2005 or include an extension schema description;

**PDF/X** The XMP identification properties for PDF/X-4/5 are created automatically.

**PDF/VT** The XMP identification properties for PDF/VT are created automatically.

**PDF/UA** The XMP identification properties for PDF/UA are created automatically.

## 14.3 Tagged PDF

The *tagged* option in `PDF_begin_document()` must be set to *true* in order to generate Tagged PDF. Tagged PDF mode is automatically activated for the PDF/A-1a/2a/3a and PDF/UA standards. It is strongly recommended to obey to PDF/UA rules when creating Tagged PDF.

*Cookbook* A full code sample can be found in the *Cookbook* topic `interchange/starter_pdfua1`.

---

C++ Java C# `int begin_item(String tagname, String optlist)`

Perl PHP `int begin_item(string tagname, string optlist)`

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

---

Open a structure element or other content element for Tagged PDF.

**tagname** Name of the item's element structure type. The following groups of element types are supported according to Table 14.3 (see PDFlib Tutorial for details):

- ▶ standard element types (a detailed description of standard element types can be found in the PDFlib Tutorial)
- ▶ pseudo element types which are not structure elements
- ▶ The tag name `Plib_custom_tag` implies use of a custom element type (this is equivalent to `customtag=true`); the actual tag name must be supplied in the *tagname* option. Custom element types require the *rolemap* document option.

The tag name can alternatively be provided via the *tagname* option which overrides this parameter.

**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 2.3)
- ▶ Tag control and accessibility options according to Table 14.4: *ActualText, Alt, customtag, E, inline, Lang, tagname, Placement, Title*
- ▶ Options related to Artifacts according to Table 14.4: *artifactssubtype, artifacttype, Attached*
- ▶ Table-related options according to Table 14.4: *ColSpan, Headers, id, RowSpan, Scope, Summary*
- ▶ Geometry options according to Table 14.4: *BBox, Height, usercoordinates, Width*
- ▶ Options for the relationship of elements according to Table 14.4: *index, parent*
- ▶ Option for attaching a bookmark according to Table 14.4: *bookmark*
- ▶ Option for specifying a list property according to Table 14.4: *ListNumbering*
- ▶ Option for inserting a nested structure element according to Table 14.4: *tag*

**Returns** An item handle which can be used in subsequent item-related calls.

**Details** Start a new structure element or Artifact (collectively called *item*). By default, the new element is inserted as a child of the currently active item. However, another position in the structure tree can be specified with the *parent* and *index* options. Structure elements can be nested to an arbitrary level. Except for pseudo and inline types structure ele-

Table 14.3 Standard, pseudo, and custom element types for `PDF_begin_item()`, `PDF_begin_mc()`, and `PDF_mc_point()` and the tag option of various functions

category	tags
<b>standard structure element types</b>	
<b>grouping</b>	Document, Part, Art, Sect, Div, BlockQuote, Caption, TOC, TOCI, Index, NonStruct, Private
<b>heading and paragraph</b>	P, H, H1, H2, H3, H4, H5, H6, H7, H8 etc. (BLSEs)
<b>label and list</b>	L, LI, Lb1, LBody (BLSEs)
<b>table</b>	Table (BLSE), TR, TH, TD, THead <sup>1</sup> , TBody <sup>1</sup> , TFoot <sup>1</sup> (all table tags can be created automatically by <code>PDF_fit_table()</code> , see PDFlib Tutorial)
<b>inline</b>	Span, Quote, Note, Reference, BibEntry, Code, Link, Annot <sup>1</sup> (ILSEs, but can be changed to BLSE with the option <code>inline=false</code> )
<b>illustration</b>	Figure, Formula, Form (ILSE)
<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> Artifact, to be distinguished from real page content.
	<b>ASpan</b> (Accessibility span; written to PDF as Span, but must be distinguished from the inline item Span) Can be used to attach accessibility attributes to content which does not belong to a structure element or which resembles a part of a structure element.
	<b>ReversedChars</b> (Not recommended) Specifies text in a right-to-left language with reversed characters.
	<b>Clip</b> (Not recommended) Specifies a marked clipping sequence, i.e. 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 <code>tagname</code> parameter. The actual tag name which will be written to PDF must be supplied in the <code>tagname</code> option. Custom element types require the <code>rolemap</code> document option.

<sup>1</sup> Requires PDF 1.5 or above

ments are not bound to the page where they have been opened, but can be continued on an arbitrary number of pages. It is recommended to avoid empty structure elements.

Structure elements and `Alt/ActualText` attributes must be properly nested according to the rules in the PDFlib Tutorial. Some decorative elements are automatically tagged as *Artifact*; see PDFlib Tutorial for details.

**PDF/A** Although Tagging is required for PDF/A-1a/2a/3a, there are no specific requirements for tag usage or nesting.

**PDF/UA** All text requires natural language specification.  
All image and graphics contents must be tagged as *Artifact* or *Figure*.  
Additional rules apply to various element types and options (see PDFlib Tutorial).

**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()` as well as for abbreviated tagging with the tag option of various functions

option	explanation
<b>ActualText</b>	(Hypertext string; not for pseudo tags except in PDF 1.5 with ASpan; if used in PDF 1.4 the inline option must be set to false) Equivalent replacement text for the content item and its kids.
<b>Alt</b>	(Hypertext string; not for pseudo tags except in PDF 1.5 with ASpan; if used in PDF 1.4 the inline option must be set to false) Word or phrase as alternate description for the content item and its children. It should be provided for figures, images, etc. which cannot be recognized as text.
<b>artifact-subtype</b>	(Keyword; only for tagname=Artifact and artifacttype=Pagination; PDF 1.7) Subtype of the artifact: Header, Footer, Watermark
<b>artifacttype</b>	(Keyword; only for tagname=Artifact) Identifies the artifact type of the content item: <b>Pagination</b> Ancillary page features such as running heads or page numbers <b>Layout</b> Typographic or design elements such as footnote rules or table shading <b>Page</b> Production aids such as cut marks and color bars <b>Background</b> (PDF 1.7) Images or colored blocks that run the entire length and/or width of the page or a structural element.
<b>Attached</b>	(Keyword list; only for tagname=Artifact and artifacttype=Pagination or Background with full-page background artifacts) Specify the edges of the page, if any, to which the artifact is logically attached. The list contains one to four of the keywords Top, Bottom, Left, and Right. Including both Left and Right or both Top and Bottom indicates a full-width or full-height artifact, respectively.
<b>BBox</b>	(Rectangle; only for tagname=Artifact, Figure, Form, Table; required for artifacttype=Background, otherwise optional, but recommended for Reflow) The element's bounding box in default coordinates (if usercoordinates=false) or user coordinates (if usercoordinates=true). PDFlib automatically creates the BBox for placed images, graphics, PDF pages (with tagname=Figure or Artifact), form fields (with tagname=Form or Artifact) and tables created by the table engine (with tagname=Table or Artifact).
<b>bookmark</b>	(Bookmark handle; not for inline and pseudo tags) Handle for a bookmark which will be associated with the structure element.
<b>ColSpan</b>	(Integer; only for tagname=TH and TD) Number of table columns spanned by a cell. Default: 1
<b>customtag</b>	(Boolean; requires the rolemap document option) If true, the element type name supplied in the tagname option is a custom element type which must be mapped to a standard element type via the rolemap document option. Default: false Setting this option to true is equivalent to providing the parameter tagname=Plib_custom_tag.
<b>E</b>	(Hypertext string; not for pseudo tags except ASpan; 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 considers the expansion text as a separate word even in the absence of explicit word breaks.
<b>Headers</b>	(List of strings; only for tagname=TH and TD; PDF 1.5) Each string in the list is the identifier of a table header cell (TH element) which is associated with the cell. The identifier(s) must have been assigned to the target cell with the id option. If multiple header cells are referenced, row headers should be listed before column headers. Within these groups headers should be ordered from most specific to most general.
<b>Height</b>	(Float; only for tagname=Figure, Form, Formula, Table, TD, TH) Height of the element in default coordinates (if usercoordinates=false) or user coordinates (if usercoordinates=true)
<b>id</b>	(String; not for pseudo elements and inline elements except Note; required for tagname=Note in PDF/UA) Assign an identifier to the element. The string must be unique among all structure elements.
<b>index</b>	(Integer; not for pseudo tags) The zero-based index at which to insert the element within the parent. Starting at this position, existing descendants of the parent are shifted upwards. Values between 0 and the current number of children can be supplied. The value -1 adds the element at the end, i.e. as the new last item. This is identical to supplying the current number of elements as index. Default: -1

Table 14.4 Options for structure and pseudo tags for `PDF_begin_item()`, `PDF_begin_mc()`, and `PDF_mc_point()` as well as for abbreviated tagging with the tag option of various functions

option	explanation
<b>inline</b>	(Boolean; only for tagname=Code, BibEntry, Note, Quote, Reference, Span) If true, the content item is written inline and no structure element is created. Default: true
<b>Lang</b>	(String; not for pseudo tags except ASpan) Language identifier for the content item in the format described in Table 3.3 for the lang option. This can be used to override the document's dominant language for individual content items. PDF/UA: the natural language must be specified with this option or with the lang option of <code>PDF_begin_document()</code> .
<b>List-Numbering</b>	(Keyword; only for tagname=L; required in PDF/UA; must be None in PDF/UA for lists where no LI child contains any Lb1 element) Numbering system used for the contents of the Lb1 elements in a numbered list or the symbol which precedes each item in an unnumbered list (default: None): <b>Circle</b> Open circular bullet <b>Decimal</b> Decimal arabic numerals (1–9, 10–99, ...) <b>Disc</b> Solid circular bullet <b>LowerAlpha</b> Lowercase letters (a, b, c, ...) <b>LowerRoman</b> Lowercase roman numerals (i, ii, iii, iv, ...) <b>None</b> No autonumbering; Lb1 elements (if present) contain arbitrary text without numbering. In this case any graphics representing the list's labels should be marked as Artifact. <b>Square</b> Solid square bullet <b>UpperAlpha</b> Uppercase letters (A, B, C, ...) <b>UpperRoman</b> Uppercase roman numerals (I, II, III, IV, ...)
<b>parent</b>	(Item handle; not for pseudo tags) The element's parent as returned by an earlier call to <code>PDF_begin_item()</code> or the activeitemid keyword of <code>PDF_get_option()</code> . The value 0 refers to the structure tree root. -1 refers to the currently active element. In other words, parent=-1 opens a child of the current element. Pseudo and inline element types are not allowed as parent. Default: -1
<b>Placement</b>	(Keyword; not for pseudo elements and the inline elements Span, Quote, Note, Reference, BibEntry, and Code) Specifies positioning of the element with respect to the enclosing reference area. This is relevant when the document is reformatted or exported to other formats. The option Placement=Block is recommended for Figure, Formula, Form, Link, Annot elements if they are created as children of grouping elements (default: Inline): <b>Before</b> Placed so that the »before« edge of the element's allocation rectangle coincides with that of the nearest enclosing reference area. <b>Block</b> Stacked in the block progression direction within an enclosing reference area or parent BLSE. <b>End</b> Placed so that the »end« edge of the element's allocation rectangle coincides with that of the nearest enclosing reference area. <b>Inline</b> Packed in the inline progression direction within an enclosing BLSE. <b>Start</b> Placed so that the »start« edge of the element's allocation rectangle coincides with that of the nearest enclosing reference area.
<b>RowSpan</b>	(Integer; only for tagname=TH and TD) The number of table rows spanned by a cell. Default: 1
<b>Scope</b>	(Keyword; only for tagname=TH; PDF 1.5; required for PDF/UA) One of the keywords Row, Column, or Both indicating whether the table 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>Summary</b>	(Hypertext string; only for tagname=Table; PDF 1.7) Summary of the table's purpose and structure
<b>tag</b>	(Option list) Additional structure element which will be inserted as a child of the current element. All options according to Table 14.4 are supported as suboptions. Tags can be nested to an arbitrary level.

Table 14.4 Options for structure and pseudo tags for `PDF_begin_item()`, `PDF_begin_mc()`, and `PDF_mc_point()` as well as for abbreviated tagging with the `tag` option of various functions

option	explanation
<b>tagname</b>	(Name string; except in <code>PDF_add_table_cell()</code> this option is required for abbreviated tagging with the <code>tag</code> option) Name of a standard element type or a pseudo element type according to Table 14.3, or name of a custom tag. The value of this option can alternatively be provided via the function parameter <code>tagname</code> .  Specify the option <code>customtag=true</code> or the parameter <code>tagname=Plib_custom_tag</code> to supply custom tag names. In this case the <code>tagname</code> option contains an 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 type names are restricted to 127 winansi characters or a sequence of Unicode characters which expands to a maximum of 127 UTF-8 bytes. Custom element type names must not start with the reserved prefix <code>Plib</code> .
<b>Title</b>	(Hypertext string; not for inline and pseudo tags; recommended for headings in PDF/UA) Title of the structure element. The title may be useful for viewing or manipulating the structure tree in Acrobat.
<b>user-coordinates</b>	(Boolean) If false, <code>BBox</code> , <code>Width</code> and <code>Height</code> are expected in default coordinates; otherwise user coordinates will be used. Default: the value of the <code>usercoordinates</code> global option
<b>Width</b>	(Float; only for <code>tagname=Figure</code> , <code>Form</code> , <code>Formula</code> , <code>Table</code> , <code>TD</code> , <code>TH</code> ) Width of the element in default coordinates (if <code>usercoordinates=false</code> ) or user coordinates (if <code>usercoordinates=true</code> )

C++ Java C# `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. An item can only be closed if all of its children have been closed before. After closing an item its parent becomes the active item.

**Scope** `page` for inline items; for grouping items also `document`; must always be paired with a matching `PDF_begin_item()` call. This function is only allowed in Tagged PDF mode.

**Abbreviated tagging.** Structure elements and artifacts can be created with `PDF_begin/end_item()` pairs. As an alternative, abbreviated tagging is available with the `tag` option of the following functions (see Table 14.5):

- ▶ `PDF_add_table_cell()` and the corresponding options in `PDF_add_table_cell()`: `fitgraphics`, `fitimage`, `fitpath`, `fitpdipage`, `fittextline`, `fittextflow`, `fitannotation`, `fitfield`
- ▶ `PDF_begin_document()`: abbreviated tagging can be used to create the root element of the structure hierarchy
- ▶ `PDF_create_annotation()`
- ▶ `PDF_create_field()`
- ▶ `PDF_draw_path()`
- ▶ `PDF_fit_graphics()`
- ▶ `PDF_fit_image()`
- ▶ `PDF_fit_pdi_page()` and `PDF_info_pdi_page()`
- ▶ `PDF_fit_table()`; abbreviated tagging triggers automatic table tagging

- ▶ `PDF_fit_textflow()`
- ▶ `PDF_fit_textline()`
- ▶ the `matchbox` option of various functions

Abbreviated tagging cannot be used to create grouping elements except when used with `PDF_begin_document()` (since all other functions create direct content, which is not allowed for grouping elements). A detailed description of abbreviated tagging can be found in the PDFlib Tutorial.

Table 14.5 Option for abbreviated tagging in `PDF_add_table_cell()` and the corresponding `fit*` options in `PDF_add_table_cell()`, `PDF_create_annotation()`, `PDF_create_field()`, `PDF_draw_path()`, `PDF_fit_graphics()`, `PDF_fit_image()`, `PDF_fit_pdi_page()`, `PDF_fit_table()`, `PDF_fit_textflow()`, `PDF_fit_textline()`, and the `matchbox` option of various functions

option	explanation
<code>tag</code>	(Option list) Create a structure element or artifact for the placed content. The suboptions listed in Table 14.4 can be used.

---

C++ Java C# **`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 items can not be activated.

**Details** Suspending 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.

While the `parent` and `index` tagging options (see Table 14.4) can be used to insert structure elements at a specific location in the structure tree, `PDF_activate_item()` can be used to add more content to a previously created structure element.

In order to work around problems in Acrobat, no direct content should be added immediately after calling `PDF_activate_item()`, but only other structure elements.

**Scope** `document, page`; this function is only allowed in Tagged PDF mode.

## 14.4 Marked Content

C++ Java C# **void begin\_mc(String tagname, String optlist)**

Perl PHP **begin\_mc(string tagname, string optlist)**

C **void PDF\_begin\_mc(PDF \*p, const char \*tagname, const char \*optlist)**

Begin a marked content sequence with optional properties.

**tagname** The name of the marked content sequence. The following tags are supported:

- ▶ All inline and pseudo tags in Table 14.3.
- ▶ The tag name *Plib\_custom* can be used for custom entries with user-defined properties.
- ▶ The tag name *Plib* is reserved.

**optlist** The following options for marked content sequences are supported:

- ▶ Options for standard properties of the marked content sequence according to Table 14.4.
- ▶ The tags *Plib\_custom* and *Plib* additionally support the option in Table 14.6.

**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 scope.

Table 14.6 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 <i>tagname=Plib</i> and <i>tagname=Plib_custom</i> ) 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 <i>type=string</i> , otherwise string; required) Value of the property

C++ Java C# **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 scope.



---

C++ Java C# **void mc\_point(String tagname, String optlist)**

Perl PHP **mc\_point(string tagname, string optlist)**

C **void PDF\_mc\_point(PDF \*p, const char \*tagname, const char \*optlist)**

---

Add a marked content point with optional properties.

**tagname** The name of the marked content point. The following tags are supported:

- ▶ All inline and pseudo tags in Table 14.3.
- ▶ The tag name *Plib\_custom* can be used for custom entries.
- ▶ The tag name *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 14.6.

**Details** A marked content point with the specified tag name 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*

## 14.5 Document Part Hierarchy

---

C++ Java C# **void begin\_dpart(String optlist)**

Perl PHP **begin\_dpart(string optlist)**

C **void PDF\_begin\_dpart(PDF \*p, const char \*optlist)**

---

Open a new node in the document part hierarchy (requires PDF/VT or PDF 2.0).

**optlist** An option list specifying document part hierarchy options according to Table 14.7: *associatedfiles*, *dpm*

PDF/VT The first call to **PDF\_begin\_dpart()** implicitly creates the root node of the document part (DPart) hierarchy. It is an error to call **PDF\_begin\_dpart()** more than once at the top level.

A call to **PDF\_begin\_dpart()** followed by a call to **PDF\_begin\_page\_ext()** defines the start of the page range of a document part. All subsequent pages until the next call to **PDF\_begin\_dpart()** belong to the same document part. All calls together create the document part hierarchy as a tree structure which can contain two types of nodes:

- ▶ Inner nodes have one or more other nodes as descendants. The descendants may be inner nodes or leaf nodes, but a particular inner node may not contain descendants of both types.
- ▶ Leaf nodes describe the page(s) in a range. Leaf nodes never have descendant nodes.

The calls to **PDF\_begin\_dpart()** and **PDF\_end\_dpart()** must match when **PDF\_end\_document()** is called. If a document part hierarchy is to be created for the document, this function must be called at least once before calling **PDF\_begin\_page\_ext()** for the first time. Calling **PDF\_begin\_dpart()** multiply creates deeper levels of the document part hierarchy. The generated depth of the document part hierarchy (i.e. the maximum nesting levels of **PDF\_begin\_dpart()**) must match the length of the list specified with the *nodenamelist* document option.

*Scope* *document*; this function must always be paired with a matching **PDF\_end\_dpart()** call.

Table 14.7 Options for **PDF\_begin/end\_dpart()**

<b>option</b>	<b>description</b>
<b>associatedfiles</b>	(List of asset handles; only for PDF 2.0 and PDF/A-3) Asset handles for associated files according to PDF/A-3. The files must have been loaded with <b>PDF_load_asset()</b> and <i>type=attachment</i> .
<b>dpm</b>	(POCA container handle; may be supplied to <b>PDF_begin_dpart()</b> or <b>PDF_end_dpart()</b> , but not to both functions for the same document part) Handle for a dictionary container created with <b>PDF_poca_new()</b> which contains document part metadata for the new node. The dictionary must have been created with the option <i>usage=dpm</i> .

---

C++ Java C# **void end\_dpart(String optlist)**

Perl PHP **end\_dpart(string optlist)**

C **void PDF\_end\_dpart(PDF \*p, const char \*optlist)**

---

Close a node in the document part hierarchy (requires PDF/VT or PDF 2.0).

**optlist** An option list specifying document part hierarchy options according to Table 14.7: *associatedfiles*, *dpm*

*PDF/VT* The first call to `PDF_end_dpart()` after `PDF_end_page_ext()` implicitly defines the end of the page range belonging to a leaf of the document part hierarchy. The calls to `PDF_begin_dpart()` and `PDF_end_dpart()` must match when `PDF_end_document()` is called.

*Scope* *document*; this function must always be paired with a matching `PDF_begin_dpart()` call.



# A List of all API 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="#">set_option</a>	<a href="#">load_font</a>	<a href="#">set_text_option</a>	<a href="#">setcolor</a>
<a href="#">get_option</a>	<a href="#">close_font</a>	<a href="#">fit_textline</a>	<a href="#">makespotcolor</a>
<a href="#">get_string</a>	<a href="#">setfont</a>	<a href="#">info_textline</a>	<a href="#">load_iccprofile</a>
<a href="#">new (C only)</a>	<a href="#">info_font</a>	<a href="#">add_textflow</a>	<a href="#">begin_pattern_ext</a>
<a href="#">delete</a>	<a href="#">begin_font</a>	<a href="#">create_textflow</a>	<a href="#">end_pattern</a>
<a href="#">create_pvf</a>	<a href="#">end_font</a>	<a href="#">fit_textflow</a>	<a href="#">shading_pattern</a>
<a href="#">delete_pvf</a>	<a href="#">begin_glyph_ext</a>	<a href="#">info_textflow</a>	<a href="#">shfill</a>
<a href="#">info_pvf</a>	<a href="#">end_glyph</a>	<a href="#">delete_textflow</a>	<a href="#">shading</a>
<a href="#">get_errnum</a>	<a href="#">encoding_set_char</a>		
<a href="#">get_errmsg</a>		<b>Table Formatting</b>	<b>Image and template</b>
<a href="#">get_apiname</a>	<b>Simple Text Output</b>	<a href="#">add_table_cell</a>	<a href="#">load_image</a>
<a href="#">get_opaque (C/C++ only)</a>	<a href="#">set_text_pos</a>	<a href="#">fit_table</a>	<a href="#">close_image</a>
<a href="#">poca_new</a>	<a href="#">show</a>	<a href="#">info_table</a>	<a href="#">fit_image</a>
<a href="#">poca_delete</a>	<a href="#">show_xy</a>	<a href="#">delete_table</a>	<a href="#">info_image</a>
<a href="#">poca_insert</a>	<a href="#">continue_text</a>		<a href="#">begin_template_ext</a>
<a href="#">poca_remove</a>	<a href="#">stringwidth</a>	<b>Matchboxes</b>	<a href="#">end_template_ext</a>
		<a href="#">info_matchbox</a>	
<b>Document and Page</b>	<b>Unicode Conversion</b>		<b>SVG Graphics</b>
<a href="#">begin_document</a>	<a href="#">convert_to_unicode</a>		<a href="#">load_graphics</a>
<a href="#">begin_document_callback (C/C++ only)</a>			<a href="#">close_graphics</a>
<a href="#">end_document</a>			<a href="#">fit_graphics</a>
<a href="#">get_buffer</a>			<a href="#">info_graphics</a>
<a href="#">begin_dpart</a>			
<a href="#">end_dpart</a>			
<a href="#">begin_page_ext</a>			
<a href="#">end_page_ext</a>			
<a href="#">suspend_page</a>			
<a href="#">resume_page</a>			
<a href="#">define_layer</a>			
<a href="#">set_layer_dependency</a>			
<a href="#">begin_layer</a>			
<a href="#">end_layer</a>			

**Graphics State***set\_graphics\_option**setlinewidth**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**elliptical\_arc**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*  
*(C/C++ only)**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**fill\_graphicsblock***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**load\_asset***Document Interchange***set\_info**begin\_item**end\_item**activate\_item**begin\_mc**end\_mc**mc\_point*

# B 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()` 101

## 3D

**3Dactivate** in `create_annotation()` 246

**3Ddata** in `create_annotation()` 246

**3Dinitialview** in `create_annotation()` 246

**3Dinteractive** in `create_annotation()` 246

**3Dshared** in `create_annotation()` 246

**3Dview** in `create_action()` 227

## A

**acrobat** in `info_font()` 72

### action

in `begin/end_page_ext()` 53

in `create_annotation()` 213

in `create_bookmark()` 209

in `create_field/group()` 221

in `end_document()` 43

in `process_pdi()` 197

**activate** suboption of `richmedia` in `create_annotation()` 250

**activeitemid** keyword in `get_option()` 29

**activeitemindex** keyword in `get_option()` 29

**activeitemisinline** keyword in `get_option()` 29

**activeitemkidcount** keyword in `get_option()` 29

**activeitemname** keyword in `get_option()` 29

**activeitemstandardname** keyword in `get_option()` 29

**actual** in `info_font()` 71

### actualtext

in `set_text_option()`, `fit_textline()`, and

`fill_textblock()` 75

**ActualText** in `begin_item()` and the tag option 260

**addfitbox** suboption for `wrap` in `fit_textflow()` 108

**addpath** keyword in `add_path_point()` 151

**adjustmethod** in `add/create_textflow()` 98

### adjustpage

in `fit_image/fit_graphics/fit_pdi_page()` 173

in `fit_pdi_page()` 194

**advancedlinebreak** in `add/create_textflow()` 98

### align

in `draw_path()` 125

keyword for the transform option in `begin_pattern_ext()` 163

**alignchar** in `fit/info_textline()` 125

### alignment

in `add/create_textflow()` 96

in `create_annotation()` 213

suboption for `leader` in `fit/info_textline()` and `add/create_textflow()` 91

**alpha** keyword for the type suboption of `softmask` in `create_gstate()` 140

**alphachannelname** in `load_image()` 169

**alphaisshape** in `create_gstate()` 140

**Alt** in `begin_item()` and the tag option 260

**angle** keyword in `info_textline()` 93

**angularunit** suboption for `georeference` 238

**animation** suboption for the `activate` suboption of `richmedia` in `create_annotation()` 251

**annotation** suboption for `targetpath` in `create_action()` 230

**annotationtype** in `add_table_cell()` and suboption for the `caption` option 115

**annotcolor** in `create_annotation()` 213

### antialias

in `shading()` 165

suboption for `shading` option of several functions 137

**api** in `info_font()` 71, 72

**apiversion** sub-suboption for `portfolio` in `PDF_add_portfolio_file/folder()` 237

**area** suboption for `fill` in `fit_table()` 118

**areaunit** suboption for `georeference` 238

**artbox** in `begin/end_page_ext()` 53

**artifactssubtype** in `begin_item()` and the tag option 260

**artifacttype** in `begin_item()` and the tag option 260

### ascender

in `info_font()` 71

in `load_font()` 65

keyword in `info_textline()` 93

**asciifile** in `set_option()` 25

### assets

sub-suboption for `portfolio` in

`PDF_add_portfolio_file/folder()` 237

suboption of `richmedia` in

`create_annotation()` 250

### associatedfiles

in `begin/end_dpart()` 266

in `begin/end_page_ext()` 53

in `end_document()` 43

in `load_image()`, `load_graphics()`,

`open_pdi_page()`, and `begin_template_ext()` 183

**Attached** in `begin_item()` and the tag option 260

**attachment** in `create_annotation()` 213  
**attachmentpassword** in `begin_document()` 47  
**attachmentpoint** in `draw_path()` 125  
**attachments** in `begin/end_document()` 43  
**autospace** in `set_option()` 25  
**autosubsetting** in `load_font()` 65  
**autoxmp** in `begin/end_document()` 43  
**avoidbreak** in `add/create_textflow()` 98  
**avoiddemostamp** in `set_option()` 25  
**avoidemptybegin** in `add/create_textflow()` 97  
**avoidwordsplitting**  
    in `add_table_cell()` 113  
    in `fit_textflow()` 104

## B

**backdropcolor** suboption of `softmask` in  
    `create_gstate()` 140  
**background** in `create_3dview()` 242  
**backgroundcolor** in `create_field/group()` 221  
**barcode** in `create_field/group()` 222  
**basestate** in `set_layer_dependency()` 60  
**BBox** in `begin_item()` and the tag option 260  
**bboxexpand** in `draw_path()` 154  
**bboxwidth**, **bboxheight** keywords in `info_path()`  
    155  
**begoptlistchar** in `create_textflow()` 102  
**beziers** suboption for `wrap` in `fit_textflow()` 108  
**bitreverse** in `load_image()` 171  
**bleedbox** in `begin/end_page_ext()` 53  
**blendmode** in `create_gstate()` 140  
**blind**  
    in `fit_table()` 117  
    in `fit_textflow()` 104  
    in many functions 125  
**block** in `process_pdi()` 198  
**blockname** suboption of `block` in `process_pdi()`  
    198  
**blocks** in `begin/end_page_ext()` 53  
**bookmark** in `begin_item()` and the tag option  
    260  
**bordercolor** in `create_field/group()` 221  
**borderstyle**  
    in `create_annotation()` 213  
    in `create_field/group()` 222  
**borderwidth** in several functions 135  
**bottom** in `add_nameddest()` and suboption for  
    destination in `create_action()`,  
    `create_annotation()`, `create_bookmark()` and  
    `begin/end_document()` 231

## boundingbox

    in `begin_glyph_ext()` 85  
    in `begin_pattern_ext()` 162  
    in `draw_path()` 154  
    in `shading()` 165  
    keyword in `info_*`() 130  
    keyword in `info_matchbox()` 134  
    keyword in `info_textflow()` 110  
    suboption for viewports option in `begin/  
    end_page_ext()` 238  
**bounds** suboption for `georeference` 238  
**boxes** suboption for `wrap` in `fit_textflow()` 108  
**boxexpand** in `open_pdi_page()` 192  
**boxheight** suboption for `matchbox` 131  
**boxlinecount** keyword in `info_textflow()` 110  
**boxsize** in various functions 126  
**boxwidth** suboption for `matchbox` 131  
**bpc** in `load_image()` 171  
**buttonlayout** in `create_field/group()` 222  
**buttonstyle** in `create_field/group()` 222

## C

**calcorder** in `create_field/group()` 222  
**calloutline** in `create_annotation()` 213  
**camerazworld** in `create_3dview()` 242  
**cameradistance** in `create_3dview()` 243  
**canonicaldate** in `create_action()` 227  
**capheight**  
    in `info_font()` 71  
    in `load_font()` 65  
    keyword in `info_textline()` 93  
**caption**  
    in `create_field/group()` 222  
    in `fit_table()` 117  
    suboption for the `barcode` option in  
    `create_field/group()` 225  
**captiondown** in `create_field/group()` 222  
**captionoffset** in `create_annotation()` 213  
**captionposition** in `create_annotation()` 213  
**captionrollover** in `create_field/group()` 222  
**cascadedflate** in `load_image()` 169  
**category** sub-suboption for `portfolio` in  
    `PDF_add_portfolio_file/folder()` 237  
**centerwindow** suboption for `viewerpreferences`  
    in `begin/end_document()` 50  
**charclass** in `add/create_textflow()` 99  
**charmapping** in `add/create_textflow()` 100  
**charref**  
    in `set_option()` 26  
    in `set_text_option()`, `fit/info_textline()`,  
    `fill_textblock()` and `add/create_textflow()` 75  
**charspacing**  
    in `create_field/group()` 222  
    in `set_text_option()`, `fit/info_textline()`,  
    `fill_textblock()` and `add/create_textflow()` 76  
**checkcolorspace**  
    keyword in `info_font()` 71  
    keyword in `info_image()` 174



**checkoutintentprofile**  
in `open_pdi_document()` 188

**checktags**  
in `begin_document()` 46  
in `open_pdi_document()` 188

**checktransgroupprofile** in `open_pdi_page()` 192

**children** in `set_layer_dependency()` 60

**cid** in `info_font()` 70, 71

**cidfont** in `info_font()` 71

**circle** keyword in `add_path_point()` 151

**circles** suboption for `wrap` in `fit_textflow()` 108

**circular** keyword in `add_path_point()` 151

**classes** suboption for logging in `set_option()` 19

**clip** in `draw_path()` 154

**clipping** suboption for `matchbox` 132

**clippingarea** in `open_pdi_page()` 192

**clippingpath** keyword in `info_image()` 174

**clippingpathname** in `load_image()` 169

**cliprule** in several functions 135

**clockwise**  
in `add_path_point()` 152  
in `elliptical_arc()` 148

**cloneboxes**  
in `fit_pdi_page()` 194  
in `open_pdi_page()` 193

**close**  
in `add_path_point()` 152  
in `draw_path()` 154

**cloudy** in `create_annotation()` 213

**code**  
in `begin_glyph_ext()` 85  
in `info_font()` 70, 71

**codepage** in `info_font()` 71

**codepagelist** in `info_font()` 71

**colorize** in `load_image()` 169

**colorized** in `begin_font()` 84

**colscalegroup** in `add_table_cell()` 113

**colspan** in `add_table_cell()` 113

**ColSpan** in `begin_item()` and the tag option 260

**colwidth** in `add_table_cell()` 113

**colwidthdefault** in `fit_table()` 117

**comb** in `create_field/group()` 222

**comment** option list macro definition in `fit_textflow()` 99

**commitonselect** in `create_field/group()` 222

**compatibility** in `begin_document()` 45

**components** in `load_image()` 171

**compress**  
in `set_option()` 26  
suboption for `metadata` 257

**condition** suboption for the `activate` suboption of `richmedia` in `create_annotation()` 251

**configuration** suboption of `richmedia` in `create_annotation()` 250

**containertype** in `poca_new()` 37

**contents** in `create_annotation()` 214

**continuetextflow** in `add_table_cell()` 113

**control** keyword in `add_path_point()` 151

**convert** in `pcos_get_stream()` 201

**copy** in `create_pvf()` 34

**copyglobals** in `load_image()` 171

**count** keyword in `info_matchbox()` 134

**coversheet** suboption for `portfolio` in `end_document()` 235

**coversheetfolder** suboption for `portfolio` in `end_document()` 235

**crease** suboption for `rendermode` in `create_3dview()` 245

**createdate** in `create_annotation()` 214

**createfittext** in `fit_textflow()` 104

**createlastindent** in `fit_textflow()` 104

**creatematchboxes** suboption for `wrap` in `fit_textflow()` 108

**createorderlist** in `set_layer_dependency()` 60

**createoutput** in `begin_document()` 48

**createpvf** in `begin_document()` 48

**createrichtext** in `create_annotation()` 214

**createtemplatein** `load_image()` 169

**createwrapbox** suboption for `matchbox` 132

**creatorinfo** in `define_layer()` 57

**cropbox** in `begin/end_page_ext()` 53

**ctm\_a/b/c/d/e/f** keywords in `get_option()` 29

**currentvalue** in `create_field/group()` 222

**currentx/y** keywords in `get_option()` 29

**curve** keyword in `add_path_point()` 151

**custom** in `create_annotation()` 214

**customtag** in `begin_item()` and the tag option 260

## D

**dasharray**  
in `add_path_point()` 152  
in `create_annotation()` 214  
in `create_field/group()` 222  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76  
in several functions 135

**dashphase**  
in `add_path_point()` 152  
in several functions 135

**dataprep** suboption for the `barcode` option in `create_field/group()` 225

**deactivate** suboption of `richmedia` in `create_annotation()` 250

**debugshow** in `fit_table()` 117

**decorationabove** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76

**defaultcmyk** in `begin_page_ext()` 53

**defaultdir** in `create_action()` 227

**defaultfontfamily** in `load_graphics()` 176, 179

**defaultfontoptions** in `load_graphics()` 176

**defaultgray** in `begin_page_ext()` 53

**defaultimageoptions** in `load_graphics()` 176

**defaultrgb** in `begin_page_ext()` 53

**defaultstate** in `define_layer()` 57

**defaultvalue** in *create\_field/group()* 223

**defaultvariant** in *set\_layer\_dependency()* 60

**defaultview** in *load\_3d()* 241

**depend** in *set\_layer\_dependency()* 60

**descender**  
in *info\_font()* 71  
in *load\_font()* 65  
keyword in *info\_textline()* 93

**description**  
in *load\_asset()* and suboption for other functions 248  
in *load\_iccprofile()* 160  
keyword in *info\_graphics()* 180  
sub-suboption for portfolio in *PDF\_add\_portfolio\_file/folder()* 237

**destination**  
in *begin/end\_document()* 43  
in *create\_action()* 227  
in *create\_annotation()* 214  
in *create\_bookmark()* 209

**destname**  
in *create\_action()* 227  
in *create\_annotation()* 214  
in *create\_bookmark()* 209  
in *end\_document()* 43  
suboption for *targetpath* in *create\_action()* 230

**devicergb** in *load\_graphics()* 176

**direct** in *poca\_insert()* 38, 39

**direction** suboption for viewerpreferences in *begin/end\_document()* 50

**disable**  
suboption for 3Dactivate in *create\_annotation()* 246  
suboption for logging in *set\_option()* 18  
suboption for shadow in *add/create\_textflow()* 77

**disablestate** suboption for 3Dactivate in *create\_annotation()* 246

**display**  
in *create\_annotation()* 214  
in *create\_field/group()* 223

**displaydoctitle** suboption for viewerpreferences in *begin/end\_document()* 50

**displaysystem** suboption for georeference 238

**documentattachment** in *load\_asset()* and suboption for other functions 248

**domain**  
in *shading()* 165  
suboption for shading option of several functions 137

**doubleadapt** suboption for matchbox 132

**doubleoffset** suboption for matchbox 132

**down** suboption for template in *create\_annotation()* 217

**downsamplemask** in *load\_image()* 169

**dpi** in *load\_image()* 126

**dpm** in *begin/end\_dpart()* 266

**drawbottom, drawleft, drawright, drawtop**  
suboptions for matchbox 132

**dropcorewidths** in *load\_font()* 65

**duplex** suboption for viewerpreferences in *begin/end\_document()* 50

**duration**  
in *begin/end\_page\_ext()* 53  
in *create\_action()* 227

## E

**E** in *begin\_item()* and the tag option 260

**ecc** suboption for the barcode option in *create\_field/group()* 225

**editable** in *create\_field/group()* 223

**ellipse** keyword in *add\_path\_point()* 151

**elliptical** keyword in *add\_path\_point()* 151

**embedding** in *load\_font()* 65

**embedprofile** in *load\_iccprofile()* 160

**enable**  
suboption for 3Dactivate in *create\_annotation()* 246  
suboption for logging in *set\_option()* 18

**enablestate** suboption for 3Dactivate in *create\_annotation()* 246

**encoding**  
in *info\_font()* 71  
in *load\_font()* 65

**Encoding** in *set\_option()* 26

**end**  
suboption for matchbox 132  
suboption for shading option of several functions 137

**endcolor** suboption for shading option of several functions 137

**endingstyles** in *create\_annotation()* 214

**endoptlistchar** in *create\_textflow()* 102

**endx, endy** keywords in *info\_textline()* 93

**entire** suboption for background in *create\_3dview()* 242

**enumeratefonts** in *set\_option()* 26

**environment** suboption for pdfvt in *load\_image()*, *load\_graphics()*, *open\_pdi\_page()* and *begin\_template\_ext()* 186

**eps** suboption for the coords and displaycoords suboptions of georeference 239

**errorconditions** in *load\_graphics()* 177

**errorpolicy** option for various functions 21

**escapesequence**  
in *set\_option()* 26  
in *set\_text\_option()*, *fit/info\_textline()*, *fill\_textblock()* and *add/create\_textflow()* 75

**exceedlimit** suboption for matchbox 132

**exchangeFillColor** in *fit\_textflow()* 105

**exchangeStrokeColors** in *fit\_textflow()* 105

**exclude** in *create\_action()* 227

**exists** keyword in *info\_matchbox()* 134

**exportable** in *create\_field/group()* 223

**exportmethod** in `create_action()` 228  
**extendo**, **extendi** in `shading()` 165  
**external** in `load_asset()` and suboption for other functions 248

## F

**facecolor** suboption for `rendermode` in `create_3dview()` 245  
**fakebold** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76  
**faked** in `info_font()` 71  
**fallbackfont** in `info_font()` 71  
**fallbackfontfamily** in `load_graphics()` 177  
**fallbackfontoptions** in `load_graphics()` 177  
**fallbackfonts** in `load_font()` 66  
**fallbackheight** in `load_graphics()` 177  
**fallbackimage** in `load_graphics()` 177  
**fallbackwidth** in `load_graphics()` 177  
**familyname**  
in `begin_font()` 84  
in `info_font()` 71  
**feature** in `info_font()` 72  
**featurelist** in `info_font()` 72  
**features** in `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 92  
**fieldlist** in `add_portfolio_folder()` 233  
**fieldname** in `add_table_cell()` and suboption for the caption option 115  
**fieldtype**  
in `add_table_cell()` and suboption for the caption option 115  
in `create_fieldgroup()` 223  
**filemode** in `begin_document()` 48  
**filename**  
in `create_action()` 228  
in `load_asset()` and suboption for other functions 249  
keyword in `info_graphics()` 180  
keyword in `info_image()` 174  
suboption for logging in `set_option()` 18  
suboption for metadata 257  
suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 185  
suboption for search in `begin/end_document()` 44  
**filenamehandling** in `set_option()` 26  
**fileselect** in `create_field/group()` 223  
**fill**  
in `add_path_point()` 152  
in `draw_path()` 154  
in `fit_table()` 118

## fillcolor

in `add_path_point()` 152  
in `create_annotation()` 215  
in `create_field/group()` 223  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76  
in several functions 135  
suboption for background in `create_3dview()` 242  
suboption for leader in `fit/info_textline()` and `add/create_textflow()` 92  
suboption for shadow in `add/create_textflow()` 77

## fillrule

in `add_path_point()` 152  
in several functions 135  
suboption for wrap in `fit_textflow()` 108

**firstbodyrow** keyword in `info_table()` 120

**firstdraw** in `fit_table()` 118

## firstlinedist

in `fit_textflow()` 105  
keyword in `info_textflow()` 110

**firstparalinedcount** keyword in `info_textflow()` 110

**fitannotation** in `add_table_cell()` and suboption for the caption option 115

**fitfield** in `add_table_cell()` 115

**fitgraphics** in `add_table_cell()` and suboption for the caption option 114

**fitheight** keyword for the type option for `add_nameddest()`, as well as for the destination option 232

**fitimage** in `add_table_cell()` and suboption for the caption option 114

## fitmethod

in `create_field/group()` 223  
in `fit_textflow()` 105  
in various functions 126  
suboption for template in `create_annotation()` 217

**fitpath** in `add_table_cell()` and suboption for the caption option 114

**fitpdipage** in `add_table_cell()` and suboption for the caption option 114

**fitrect** keyword for the type option for `add_nameddest()`, as well as for the destination option 232

**fitscalex**, **fitscaley** keywords in `info_*`() 130

**fittext** keyword in `info_textflow()` 110

**fittextflow** in `add_table_cell()` and suboption for the caption option 114

**fittextline** in `add_table_cell()` and suboption for the caption option 115

## fittingpossible

keyword in `info_graphics()` 180  
keyword in `info_pdi_page()` 196

**fitvisible** keyword for the type option for `add_nameddest()`, as well as for the destination option 232

**fitvisibleheight**, **fitvisiblewidth** keywords for the type option for `add_nameddest()`, as well as for the destination option 232

**fitwidth** keyword for the type option for `add_nameddest()`, as well as for the destination option 232

#### **fitwindow**

keyword for the type option for `add_nameddest()`, as well as for the destination option 232  
suboption for viewerpreferences in `begin/end_document()` 50

**fixed** keyword for the type option for `add_nameddest()`, as well as for the destination option 232

**fixedleading** in `add/create_textflow()` 97

**fixedtextformat** in `create_textflow()` 102

**flash** sub-suboption for portfolio in `PDF_add_portfolio_file/folder()` 237

#### **flatness**

in `add_path_point()` 152  
in `create_gstate()` 139  
in several functions 135

#### **flush**

in `begin_document()` 48  
suboption for logging in `set_option()` 18

#### **font**

in `create_annotation()` 215  
in `create_field/group()` 223  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76  
suboption for leader in `fit/info_textline()` and `add/create_textflow()` 92

**FontAFM** in `set_option()` 26

**fontfile** in `info_font()` 72

#### **fontname**

in `info_font()` 72  
in `load_font()` 66

**FontnameAlias** in `set_option()` 26

**FontOutline** in `set_option()` 26

**FontPFM** in `set_option()` 27

#### **fontscale**

in `fit_textflow()` 105  
keyword in `info_textflow()` 110

#### **fontsize**

in `create_annotation()` 215  
in `create_field/group()` 223  
in `info_font()` 71  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77  
suboption for leader in `fit/info_textline()` and `add/create_textflow()` 92

#### **fontstyle**

in `create_bookmark()` 209  
in `info_font()` 72  
in `load_font()` 66

**fonttype** in `info_font()` 72

**footer** in `fit_table()` 118

**forcebox** in `open_pdi_page()` 193

**forcedheight/forcedwidth** in `load_graphics()` 177

**full** in `info_font()` 72

**functionname** in `create_action()` 228

## G

#### **georeference**

in `load_image()` 183  
suboption for viewports in `begin/end_page_ext()` 238

#### **glyphcheck**

in `set_option()` 27  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 75

**glyphid** in `info_font()` 70, 72

#### **glyphname**

in `begin_glyph_ext()` 85  
in `info_font()` 70, 72

**graphics** in `add_table_cell()` and suboption for the caption option 115

**graphicsheight**, **graphicswidth** keywords in `info_graphics()` 180

#### **group**

in `begin_page_ext()` 53  
in `resume_page()` 56  
in `set_layer_dependency()` 60  
option in `add_nameddest()` and suboption for destination option in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 231  
suboption for labels in `begin_document()` 49

**groups** in `begin_document()` 43

#### **gstate**

in `add_path_point()` 152  
in `fit_image/fit_graphics/pdi_page()` 173  
in `fit_pdi_page()` 194  
in `fit_table()` 118  
in `fit/info_textline()` and `add/create_textflow()` 105  
in many graphics functions 136  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77  
in `shading_pattern()` 164  
suboption for shadow in `add/create_textflow()` 77

## H

**header** in `fit_table()` 118

**Headers** in `begin_item()` and the tag option 260

#### **height**

in `add_path_point()` 152  
in `begin/end_page_ext()` 53  
in `load_image()` 171  
keyword in `info_*` 130  
keyword in `info_matchbox()` 134

**Height** in `begin_item()` and the tag option 260

**hide** in `create_action()` 228

**hidemenubar** suboption for viewerpreferences in `begin/end_document()` 50

**hidetoolbar** suboption for viewerpreferences in `begin/end_document()` 50

**hidewindowui** suboption for viewerpreferences in `begin/end_document()` 50

**highlight**  
in `create_annotation()` 215  
in `create_field/group()` 223

**honorclippingpath** in `load_image()` 169

**honoriccprofile** in `load_image()` 169

**horboxgap** keyword in `info_table()` 120

**horizscaling** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77

**horshrinking** keyword in `info_table()` 120

**horshrinklimit** in `fit_table()` 118

**hortabmethod** in `add/create_textflow()` 97

**hortabsize** in `add/create_textflow()` 97

**hostfont** in `info_font()` 72

**HostFont** in `set_option()` 27

**hypertextencoding**  
in `set_option()` 27  
suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 49  
suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 185  
suboption for viewpoints in `begin/end_page_ext()` 238

**hypertextformat** in `set_option()` 27

**hyphenchar** in `add/create_textflow()` 100

## I

**icomponents** keyword in `get_option()` 29

**iccprofile**  
in `get_option()` 30  
in `load_image()` 169  
keyword in `info_image()` 174

**ICCProfile** in `set_option()` 27

**iccprofilecymk**, **iccprofilegray**, **iccprofilergb** in `set_option()` 27

**icon**  
in `create_field/group()` 223  
sub-suboption for portfolio in `PDF_add_portfolio_file/folder()` 237

**icondown** in `create_field/group()` 223

**iconname**  
in `create_annotation()` 215  
in `load_image()`, `load_graphics()`, `open_pdi_page()` and `begin_template_ext()` 183

**iconrollover** in `create_field/group()` 223

**id**  
in `begin_item()` and the tag option 260  
sub-suboption for portfolio in `PDF_add_portfolio_file/folder()` 237

**ignoreclippingpath** in `fit_image()` 173

**ignoremask** in `load_image()` 169

**ignoreorientation**  
in `fit_image()` 173  
in `load_image()` 169

**ignorepdfversion** in `open_pdi_document()` 193

**image** in `add_table_cell()` and suboption for the caption option 115

**imagehandle** in `load_image()` 171

**imageheight** keyword in `info_image()` 174

**imagemask** keyword in `info_image()` 174

**imagetype** keyword in `info_image()` 174

**imagewidth** keyword in `info_image()` 174

**includeid**  
suboption for logging in `set_option()` 18

**includepid**  
suboption for logging in `set_option()` 18

**includetid**  
suboption for logging in `set_option()` 18

**index**  
in `begin_item()` and the tag option 260  
in `create_bookmark()` 210  
in `info_font()` 73  
in `info_pdi_page()` 196  
in `poca_insert()` and `poca_remove()` 39

**indextype** suboption for search in `begin/end_document()` 44

**infomode**  
in `load_image()` 169  
in `open_pdi_document()` 188  
keyword in `info_image()` 174

**initialexportstate** in `define_layer()` 57

**initialprintstate** in `define_layer()` 57

**initialsubset** in `load_font()` 66

**initialview** suboption for portfolio in `end_document()` 235

**initialviewstate** in `define_layer()` 58

**inittextstate**  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77  
in `set_text_state()` 136

**inline**  
in `begin_item()` and the tag option 261  
in `load_image()` 171

**inmemory**  
in `begin_document()` 48  
in `open_pdi_document` 188

**innerbox** suboption for matchbox 132

**inputencoding** suboption for metadata 257

**inputformat** suboption for metadata 257

**inreplyto** in `create_annotation()` 215

**instance** in `create_action()` 228

**intent** in `define_layer()` 58

**interiorcolor** in `create_annotation()` 215

**interpolate** in `load_image()` 169

**inversefill** suboption for wrap in `fit_textflow()` 108

**invert** in `load_image()` 169

**invisiblelayers** in `set_layer_dependency()` 60

**ismap** in `create_action()` 228

**istemplate** keyword in `info_graphics()` 180

**italicangle**

in `info_font()` 72

in `set_text_option()`, `fit/info_textline()`,

`fill_textblock()` and `add/create_textflow()` 77

**item** in `create_bookmark()` 210

**itemname** in `create_field/group()` 223

**itemnamelist** in `create_field/group()` 224

**itemtextlist** in `create_field/group()` 224

## J

**justifymethod** in `fit/info_textline()` 90

## K

**K** in `load_image()` 171

**keepfilter** in `pcos_get_stream()` 201

**keepfont** in `load_font()` 66

**keephandles** in `delete_table()` 121

**keepnative**

in `info_font()` 72

in `load_font()` 66

**keepxmp** suboption for metadata 257

**kerning**

in `set_option()` 27

in `set_text_option()`, `fit/info_textline()`,

`fill_textblock()` and `add/create_textflow()` 77

**kerningpairs** in `info_font()` 72

**key**

in `poca_insert()` and `poca_remove()` 39

suboption for custom in `create_annotation()` 214

suboption for fieldlist in `add_portfolio_file/folder()` 235

suboption for properties in `begin_mc()` and `mc_point()` 264

## L

**label** in `begin/end_page_ext()` 53

**labels** in `begin/end_document()` 43

**lang**

in `begin_document()` 46

in `load_graphics()` 177

keyword in `info_pdi_page()` 196

**Lang** in `begin_item()` and the tag option 261

**language**

in `define_layer()` 58

in `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 92

in `info_font()` 72

**largearc**

in `add_path_point()` 152

in `elliptical_arc()` 148

**lastalignment** in `add/create_textflow()` 97

**lastbodyrow** keyword in `info_table()` 120

**lastfont** keyword in `info_textflow()` 110

**lastfontsize** keyword in `info_textflow()` 110

**lastlinedist**

in `fit_textflow()` 105

keyword in `info_textflow()` 110

**lastmark** keyword in `info_textflow()` 110

**lastparalinecount** keyword in `info_textflow()` 110

**layer**

in `create_annotation()` 215

in `create_field/group()` 224

in `load_image()`, `load_graphics()`,

`open_pdi_page()` and `begin_template_ext()` 183

**layerstate** in `create_action()` 228

**leader**

in `add/create_textflow()` 97

in `fit/info_textline()` 90

**leaderlength** in `create_annotation()` 215

**leaderoffset**

in `create_annotation()` 215

**leading**

in `set_text_option()`, `fit/info_textline()`,

`fill_textblock()` and `add/create_textflow()` 77

keyword in `info_textflow()` 110

**left** option in `add_nameddest()` and suboption

for destination in `create_action()`,

`create_annotation()`, `create_bookmark()` and `begin/end_document()` 231

**lefttindint** in `add/create_textflow()` 97

**leftindex**, **leftliney** keywords in `info_textflow()` 110

**license** in `set_option()` 27

**licensefile** in `set_option()` 27

**lighting** in `create_3dview()` 243

**limitcheck** in `begin_document()` 45

**line**

in `create_annotation()` 216

keyword in `add_path_point()` 151

suboption for stroke in `fit_table()` 119

**linearize** in `begin_document()` 43

**linearunit** suboption for georeference 239

**linecap**

in `add_path_point()` 152

in `create_gstate()` 139

in several functions 136

**linegap**

in `info_font()` 72

in `load_font()` 66

**lineheight** suboption for wrap in `fit_textflow()` 108

**linejoin**

in `add_path_point()` 152

in `create_gstate()` 139

in several functions 136

**linespreadlimit** in `fit_textflow()` 105

## **linewidth**

- in `add_path_point()` 152
- in `create_annotation()` 216
- in `create_field/group()` 224
- in `create_gstate()` 139
- in several functions 136

**listmode** in `set_layer_dependency()` 60

**ListNumbering** in `begin_item()` and the tag option 261

**loadtype** sub-suboption for portfolio in `PDF_add_portfolio_file/folder()` 237

## **locale**

- in `add/create_textflow()` 98
- sub-suboption for portfolio in `PDF_add_portfolio_file/folder()` 237

## **locked**

- in `create_annotation()` 216
- in `create_field/group()` 224

**lockedcontents** in `create_annotation()` 216

**lockmode** in `create_field/group()` 224

**logging** in `set_option()` 27

**luminosity** keyword for the type suboption of `softmask` in `create_gstate()` 140

# **M**

**macro** option list macro definition in `fit_textflow()` 101

**maingid** in `info_font()` 72

**major** keyword in `get_option()` 29

**mappoints** suboption for georeference 239

**mapsystem** suboption for georeference 239

## **margin**

- in `add_table_cell()` 113
- in `fit_textline()` 126
- suboption for `matchbox` 132

**marginbottom, marginleft, marginright, margintop** in `add_table_cell()` 113

**mark** in `add/create_textflow()` 99

**mask** in `load_image()` 170

**masked** in `load_image()` 170

**masterpassword** in `begin_document()` 47

## **matchbox**

- in `add_table_cell()` and suboption for the caption option 115
- in `add/create_textflow()` 99
- in various functions 126
- suboption for `createlastindent` in `fit_textflow()` 104

**matrix** keyword for the transform option in `begin_pattern_ext()` 163

**maxchar** in `create_field/group()` 224

**maxcode** in `info_font()` 72

**maxfilehandles** in `set_option()` 27

**maxlinelength** keyword in `info_textflow()` 110

**maxlines** in `fit_textflow()` 105

**maxliney** keyword in `info_textflow()` 110

**maxspacing** in `add/create_textflow()` 98

**maxvusunicode** in `info_font()` 72

**mediabox** in `begin/end_page_ext()` 53

**menuname** in `create_action()` 228

## **metadata** 257

- in `begin/end_document()` 43
- in `begin/end_page_ext()` 54
- in `load_font()` 66
- in `load_iccprofile()` 160
- in `load_image()`, `load_graphics()`, `open_pdi_page()` and `begin_template_ext()` 183
- keyword in `info_graphics()` 180

**metricsfile** in `info_font()` 72

**mimetype** in `load_asset()` and suboption for other functions 249

**minfontsize** in `fit_textflow()` 106, 126

**mingapwidth** in `fit_textflow()` 106

**minlinecount** in `add/create_textflow()` 97

**minlinelength** keyword in `info_textflow()` 110

**minliney** keyword in `info_textflow()` 110

**minor** keyword in `get_option()` 29

**minrowheight** in `add_table_cell()` 113

**minspacing** in `add/create_textflow()` 98

**minvusunicode** in `info_font()` 73

## **mirroringx, mirroringy**

- keywords in `info_image()` 174
- keywords in `info_pdi_page()` 196

## **miterlimit**

- in `add_path_point()` 152
- in `create_gstate()` 139
- in several functions 136

**moddate** in `begin/end_document()` 43

**modeltree** suboption for `3Dactivate` in `create_annotation()` 246

**move** keyword in `add_path_point()` 151

**movieposter** in `create_annotation()` 216

**multiline** in `create_field/group()` 224

**multiselect** in `create_field/group()` 224

# **N**

## **N**

- in `shading()` 165
- suboption for shading option of several functions 137

## **name**

- in `add_path_point()` 152
- in `create_3dview()` 243
- in `create_annotation()` 216
- in `info_font()` 71, 72
- in `load_asset()` and suboption for other functions 249
- keyword in `info_matchbox()` 134
- sub-suboption for portfolio in `PDF_add_portfolio_file/folder()` 237
- suboption for `matchbox` 132
- suboption for `targetpath` in `create_action()` 230
- suboption for viewports in `begin/end_page_ext()` 238

**namelist** in `create_action()` 228  
**navigator** suboption for `portfolio` in `end_document()` 235  
**newwindow** in `create_action()` 229  
**nextline** in `add/create_textflow()` 99  
**nextparagraph** in `add/create_textflow()` 99  
**nodenamelist** in `begin_document()` 45  
**nofitlimit** in `add/create_textflow()` 98  
**nonfullscreenpagemode** suboption for `viewerpreferences` in `begin/end_document()` 50  
**normal** suboption for `template` in `create_annotation()` 217  
**normalize**  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76  
**numcids** in `info_font()` 73  
**numcopies** suboption for `viewerpreferences` in `begin/end_document()` 50  
**numglyphs** in `info_font()` 73  
**numpoints** keyword in `info_path()` 155  
**numusableglyphs** in `info_font()` 73  
**numusedglyphs** in `info_font()` 73

## O

**objectheight, objectwidth** keywords in `info_*`() 130  
**objectstreams** in `begin_document()` 44  
**offset**  
suboption for `shadow` in `add/create_textflow()` 77  
suboption for `wrap` in `fit_textflow()` 108  
**offsetbottom, offsetleft, offsetright, offsettop**  
suboptions for `matchbox` 133  
**onpanel** in `define_layer()` 58  
**opacity**  
in `create_annotation()` 216  
suboption for `rendermode` in `create_3dview()` 245  
**opacityfill, opacitystroke** in `create_gstate()` 140  
**open**  
in `create_annotation()` 216  
in `create_bookmark()` 210  
**openmode** in `begin/end_document()` 44  
**openrect** suboption for `matchbox` 133  
**operation** in `create_action()` 229  
**OPI-1.3, OPI-2.0** in `load_image()`, `load_graphics()`, and `begin_template_ext()` 183  
**optimize** in `begin_document()` 44  
**optimizeinvisible** in `load_font()` 67  
**orientate**  
in `create_annotation()` 216  
in `create_field/group()` 224  
in `fit_textflow()` 106  
in various functions 126  
**outlineformat** in `info_font()` 73  
**outputblockname** suboption of `block` in `process_pdi()` 198

**overline** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77  
**overprintfill, overprintmode, overprintstroke** in `create_gstate()` 140

## P

**page**  
in `load_image()` 170  
suboption in `add_nameddest()` and suboption for `destination` in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 231  
**pageelement** in `define_layer()` 58  
**pageheight, pagewidth**  
keywords in `get_option()` 29  
keywords in `info_pdi_page()` 196  
**pagelabel** suboption for `reference` in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 185  
**pagelayout** in `begin/end_document()` 44  
**pagenumber**  
in `begin_page_ext()` 54  
in `resume_page()` 56  
suboption for `labels` in `begin/end_document()` and `label` in `begin/end_page_ext()` 49  
suboption for `reference` in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 185  
suboption for `targetpath` in `create_action()` 230  
suboption of `block` in `process_pdi()` 198  
**pages** suboption for `separationinfo` in `begin/end_page_ext()` 54  
**painttype** in `begin_page_ext()` 162  
**parameters** in `create_action()` 229  
**parent**  
in `begin_item()` and the tag option 261  
in `create_bookmark()` 210  
in `set_layer_dependency()` 60  
**parentlayer** in `open_pdi_document` 188  
**parentname** in `create_annotation()` 216  
**parenttitle** in `open_pdi_document` 188  
**parindent** in `add/create_textflow()` 97  
**passthrough** in `load_image()` 170  
**password**  
in `create_field/group()` 224  
in `load_asset()` and suboption for other functions 249  
in `open_pdi_document` 188  
**path**  
in `add_path_point()` 152  
in `add_table_cell()` and suboption for the `caption` option 115  
suboption for `textpath` in `fit_textline()` 91  
**pathlength** keyword in `info_textline()` 93  
**pathref** keyword in `add_path_point()` 151  
**paths** suboption for `wrap` in `fit_textflow()` 108



**pdfa** in `begin_document()` 45  
**pdfua** in `begin_document()` 45  
**pdfvt**  
  in `begin_document()` 45  
  in `load_image()`, `load_graphics()`,  
  `open_pdi_page()` and `begin_template_ext()`  
  183  
**pdfx** in `begin_document()` 46  
**pdi** keyword in `get_option()` 29  
**pdiPAGE** in `add_table_cell()` and suboption for the  
  caption option 115  
**pdiusebox**  
  in `open_pdi_page()` 193  
  suboption for reference in  
  `begin_template_ext()`, `load_image()`, and  
  `open_pdi_page()` 185  
**permissions** in `begin_document()` 47  
**perpendiculardir** keyword in `info_textline()` 93  
**picktraybypdfsize** suboption for  
  viewerpreferences in `begin/end_document()`  
  50  
**Placement** in `begin_item()` and the tag option 261  
**playmode** in `create_annotation()` 216  
**polar** in `add_path_point()` 153  
**polygons** suboption for `wrap` in `fit_textflow()` 108  
**polylinelist** in `create_annotation()` 216  
**popup** in `create_annotation()` 216  
**portfolio** in `end_document()` 44  
**position**  
  in `create_field/group()` 224  
  in various functions 127  
  suboption for template in  
  `create_annotation()` 217  
**postscript** in `begin_template_ext()` 182  
**predefcmap** in `info_font()` 73  
**prefix**  
  suboption for `fieldlist` in `add_portfolio_file/`  
  `folder()` 235  
  suboption for labels in `begin/`  
  `end_document()` and label in `begin/`  
  `end_page_ext()` 49  
**presentation** suboption for the activate  
  suboption of richmedia in  
  `create_annotation()` 251  
**preserveoldpantoneNames** in `set_option()` 157  
**preservePua** in `load_font()` 67  
**preserveradio** in `create_action()` 229  
**printarea** suboption for viewerpreferences in  
  `begin/end_document()` 50  
**printclip** suboption for viewerpreferences in  
  `begin/end_document()` 50  
**printpagerange** suboption for viewerpreferences  
  in `begin/end_document()` 50  
**printscaling** suboption for viewerpreferences in  
  `begin/end_document()` 50  
**printsubtype** in `define_layer()` 58  
**properties** in `begin_mc()` and `mc_point()` 264  
**px**, **py** keywords in `info_path()` 155

## R

**ro**, **r1** in `shading()` 166  
**radians** in `add_path_point()` 153  
**radius** in `add_path_point()` 153  
**readfeatures** in `load_font()` 67  
**readkerning** in `load_font()` 67  
**readonly**  
  in `create_annotation()` 216  
  in `create_field/group()` 224  
**readselectors** in `load_font()` 67  
**readshaping** in `load_font()` 67  
**recordlevel** in `begin_document()` 46  
**recordsize** in `begin_document()` 48  
**rect** keyword in `add_path_point()` 151  
**rectangle** keyword in `info_matchbox()` 134  
**rectify**  
  in `add_path_point()` 153  
  in `elliptical_arc()` 148  
**reference** in `begin_template_ext()`,  
  `load_graphics()`, and `open_pdi_page()` 184  
**refpoint**  
  in `fill_*block()` and `info_path()` 127  
  in `info_path()` 155  
**relation** suboption for `targetpath` in  
  `create_action()` 230  
**relationship** in `load_asset()` and suboption for  
  other functions 249  
**relative** in `add_path_point()` 153  
**remove** suboption for logging in `set_option()` 18  
**removeonsuccess** suboption for logging in  
  `set_option()` 18  
**removeunused** in `define_layer()` 58  
**rendercolor** suboption for `rendermode` in  
  `create_3dview()` 245  
**renderingintent**  
  in `create_gstate()` 140  
  in `load_image()` 170  
**rendermode** in `create_3dview()` 243  
**repair** in `open_pdi_document` 188  
**repeatcontent** in `add_table_cell()` 113  
**replacedchars** in `info_textline()` 93  
**replacementchar**  
  in `info_font()` 73  
  in `load_font()` 67  
**replyto** in `create_annotation()` 217  
**required** in `create_field/group()` 224  
**requiredmode** in `open_pdi_document` 189  
**resetfont** in `add/create_textflow()` 99  
**resolution** suboption for the barcode option in  
  `create_field/group()` 225  
**resourcefile** in `set_option()` 27  
**resourcenumber** in `get_option()` 30  
**restore** in `add/create_textflow()` 99  
**resx**, **resy** keywords in `info_image()` 174  
**return**  
  in `add_table_cell()` 114  
  in `add/create_textflow()` 99  
**returnatmark** in `fit_textflow()` 106

## **returnreason**

- keyword in `info_table()` 120
- keyword in `info_textflow()` 110
- revision** keyword in `get_option()` 29
- rewind**
  - in `fit_table()` 118
  - in `fit_textflow()` 106
- richmedia** in `create_annotation()` 217
- richmediaargs** in `create_action()` 229
- richtext** in `create_field/group()` 224
- right** option in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 231
- rightindent**
  - in `add/create_textflow()` 97
  - suboption for `createlastindent` in `fit_textflow()` 104
- rightlinex**, **rightliney** keywords in `info_textflow()` 110
- righttoleft** in `info_textline()` 94
- rolemap** in `begin_document()` 46
- rollover** suboption for `template` in `create_annotation()` 217
- rotate**
  - in `begin/end_page_ext()` 54
  - in `create_annotation()` 217
  - in `fit_textflow()` 106
  - in various functions 127
  - keyword for the transform option in `begin_pattern_ext()` 163
  - keyword in `info_pdi_page()` 196
- round**
  - in `add_path_point()` 153
  - in `draw_path()` 154
  - suboption for `matchbox` 133
- rowcount** keyword in `info_table()` 120
- rowheight** in `add_table_cell()` 114
- rowheightdefault** in `fit_table()` 119
- rowjoiningroup** in `add_table_cell()` 114
- rowscalegroup** in `add_table_cell()` 114
- rowspan** in `add_table_cell()` 114
- RowSpan** in `begin_item()` and the tag option 261
- rowsplit** keyword in `info_table()` 120
- ruler** in `add/create_textflow()` 97

## **S**

- save** in `add/create_textflow()` 99
- saveresources** in `set_option()` 27
- scale**
  - in various functions 127
  - keyword for the transform option in `begin_pattern_ext()` 163
- scalex**, **scaley** keywords in `info_textline()` 94
- schema** suboption for `portfolio` in `end_document()` 236

## **scope**

- keyword in `get_option()` 29
- suboption for `pdfvt` in `load_image()`, `load_graphics()`, `open_pdi_page()` and `begin_template_ext()` 186
- Scope** in `begin_item()` and the tag option 261
- script**
  - in `create_action()` 229
  - in `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 92
  - in `info_font()` 72
  - in `load_3d()` 241
- scriptlist** keyword in `info_textline()` 94
- scriptname** in `create_action()` 229
- scripts** suboption for the `activate` suboption of `richmedia` in `create_annotation()` 251
- scrollable** in `create_field/group()` 224
- search** in `begin/end_document()` 44
- searchpath** in `set_option()` 27
- selector**
  - in `info_font()` 70
  - keyword in `info_font()` 73
- selectorlist** keyword in `info_font()` 73
- separationinfo** in `begin_page_ext()` 54
- shading** in several functions 136
- shadow** in `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77
- shaping** in `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 92
- shapingsupport** in `info_font()` 73
- showborder**
  - in `fit_textflow()` 106
  - in various functions 128
- showcaption** in `create_annotation()` 217
- showcells** in `fit_table()` 119
- showcontrols** in `create_annotation()` 217
- showgrid** in `fit_table()` 119
- showtabs** in `fit_textflow()` 106
- shrinklimit**
  - in `add/create_textflow()` 98
  - in `fit_textline()` 128
- shrug** in `open_pdi_document` 189
- shutdownstrategy** in `set_option()` 27
- simplefont**
  - in `load_font()` 67
- singfont** in `info_font()` 73
- skew** keyword for the transform option in `begin_pattern_ext()` 163
- skipembedding** in `load_font()` 68
- smoothness** in `create_gstate()` 140
- softmask** in `create_gstate()` 140
- sort** suboption for `portfolio` in `end_document()` 236
- sorted** in `create_field/group()` 224
- soundvolume** in `create_annotation()` 217
- space** in `add/create_textflow()` 99
- spellcheck** in `create_field/group()` 224

**split**  
 keyword in `info_textflow()` 110  
 suboption for `portfolio` in `end_document()` 236

**spotcolor** suboption for `separationinfo` in `begin/end_page_ext()` 54

**spotcolorlookup** in `set_option()` 157

**spotname** suboption for `separationinfo` in `begin/end_page_ext()` 54

**spreadlimit** in `add/create_textflow()` 98

**stamp**  
 in `fit_textflow()` 106  
 in various functions 127

**standardfont** in `info_font()` 73

**start**  
 suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 49  
 suboption for shading option of several functions 137

**startcolor** in `shading()` 166

**startoffset** suboption for `textpath` in `fit_textline()` 91

**startx, starty** keywords in `info_textline()` 94

**stretch** in `begin_font()` 84

**strikeout** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77

**stringformat**  
 in `set_option()` 28

**stringlimit** suboption for logging in `set_option()` 18

**strings** sub-suboption for `portfolio` in `PDF_add_portfolio_file/folder()` 237

**strips** keyword in `info_image()` 174

**stroke**  
 in `add_path_point()` 153  
 in `draw_path()` 154  
 in `fit_table()` 119

**strokeadjust** in `create_gstate()` 140

**strokecolor**  
 in `add_path_point()` 152  
 in `create_field/group()` 224  
 in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 77  
 in several functions 136  
 suboption for shadow in `add/create_textflow()` 77

**strokewidth**  
 in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 78  
 suboption for shadow in `add/create_textflow()` 77

**strongref** suboption for reference in `begin_template_ext()` and `open_pdi_page()` 185

**structuretype** in `begin_document()` 47

**style** suboption for labels in `begin/end_document()` and label in `begin/end_page_ext()` 49

**subject** in `create_annotation()` 217

**submitemptyfields** in `create_action()` 229

**submitname** in `create_field/group()` 225

**subpaths**  
 in `draw_path()` 154

**subsetlimit** in `load_font()` 68

**subsetminsize** in `load_font()` 68

**subsetting** in `load_font()` 68

**Summary** in `begin_item()` and the tag option 261

**supplement** in `info_font()` 73

**svgpath** in `add_path_point()` 153

**symbolfont** in `info_font()` 73

**symbology** suboption for the barcode option in `create_field/group()` 225

## T

**tabalignchar** in `add/create_textflow()` 100

**tabalignment** in `add/create_textflow()` 97

**tableheight, tablewidth** keywords in `info_table()` 120

**taborder**  
 in `begin/end_page_ext()` 54  
 in `create_field/group()` 225

**tag**  
 in `begin_document()` 47  
 in `begin_item()` and the tag option 261  
 in `fit_image()`, `fit_pdi_page()`, `fit_graphics()`, `fit_textline()`, `fit_textflow()`, `draw_path()`, `create_annotation()`, `fill_textblock()`, `create_field()` and suboption in `add_table_cell()` 263

**tagged** in `begin_document()` 47

**tagname** in `begin_item()` and the tag option 262

**tagtrailinghyphen** in `set_text_option()`, `fit/info_textline()` and `fill_textblock()` 78

**target**  
 in `create_action()` 229  
 suboption for reference in `begin_template_ext()`, `load_image()`, and `open_pdi_page()` 185

**targetpath**  
 in `create_action()` 229  
 suboption for `targetpath` in `create_action()` 230

**tempdirname** in `begin_document()` 48

**tempfilenames** in `begin_document()` 48

**template**  
 in `create_annotation()` 217  
 suboption of `softmask` in `create_gstate()` 140

**templateoptions** in `load_graphics()` 178

**text**  
 in `add_table_cell()` and suboption for the caption option 115  
 suboption for leader in `fit/info_textline()` and `add/create_textflow()` 92

**textcolor** in `create_bookmark()` 210

**textendx, textendy** keywords in `info_textflow()` 110

**textflow**  
in `add_table_cell()` and suboption for the caption option 115  
in `fill_textblock()` 205  
suboption for `createrichtext` in `create_annotation()` 214

**textflowhandle** in `fill_textblock()` 205

**textformat**  
in `set_option()` 28  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 76

**textheight**  
keyword in `info_textflow()` 110  
keyword in `info_textline()` 94

**textknockout** in `create_gstate()` 140

**textlen** in `create_textflow()` 102

**textpath** in `fit/info_textline()` 90

**textrendering**  
in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 78  
suboption for shadow in `add/create_textflow()` 77

**textrise** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 78

**textstate** in `get_option()` 30

**textwidth**  
keyword in `info_textflow()` 111  
keyword in `info_textline()` 94

**textx, texty** keywords in `get_option()` 29

**thumbnail** in `load_asset()` and suboption for other functions 249

**tilingtype** in `begin_page_ext()` 163

**title**  
in `create_annotation()` 217  
keyword in `info_graphics()` 180

**Title** in `begin_item()` and the tag option 262

**toggle** in `create_fieldgroup()` 225

**tolerance** suboption for `textpath` in `fit_textline()` 91

**toolbar** suboption for `3Dactivate` in `create_annotation()` 246

**tooltip** in `create_field/group()` 225

**top** option in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 231

**topdown**  
in `begin_page_ext()` 54  
in `begin_pattern_ext()` 163  
in `begin_template_ext()` 181

**topindex** in `create_field/group()` 225

**topeleveltag** keyword in `info_pdi_page()` 196

**topeleveltagcount** keyword in `info_pdi_page()` 196

**transform** in `begin_pattern_ext()` 163

**transition**  
in `begin/end_page_ext()` 54  
in `create_action()` 230

**translate** keyword for the transform option in `begin_pattern_ext()` 163

**transparencygroup**  
in `begin/end_page_ext()` 55  
in `open_pdi_page()`, `load_graphics()`, and `begin_template_ext()` 184

**transparent** keyword in `info_image()` 174

**trimbox** in `begin/end_page_ext()` 55

**truncatetrailingwhitespace** in `fit_textflow()` 106

**type**  
in `create_3dview()` 243  
in `load_3d()` 242  
in `poca_insert()` 39  
keyword in `info_graphics()` 180  
option in `add_nameddest()` and suboption for destination in `create_action()`, `create_annotation()`, `create_bookmark()` and `begin/end_document()` 232  
suboption for custom in `create_annotation()` 214  
suboption for fieldlist in `add_portfolio_file/folder()` 235  
suboption for properties in `begin_mc()` and `mc_point()` 264  
suboption for rendermode in `create_3dview()` 245  
suboption for shading option of several functions 137  
suboption for the coords and displaycoords suboptions of `georeference` 239  
suboption of `softmask` in `create_gstate()` 140

## U

**U3Dpath** in `create_3dview()` 243

**underline, underlineposition, underlinewidth** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 78

**unicode** in `info_font()` 70, 73

**unicodelfont** in `info_font()` 73

**unicodemap** in `load_font()` 68

**unisonselect** in `create_fieldgroup()` 225

**unknownchars** in `info_textline()` 94

**unmappedchars**  
in `info_font()` 73  
in `info_textline()` 94

**uri** in `begin/end_document()` 45

**url** in `create_action()` 230

**urls** in `load_iccprofile()` 160

**usage**  
in `load_iccprofile()` 160  
in `poca_new()` 38

**used** keyword in `info_textflow()` 111

**usedglyph** in `info_font()` 73

**useembeddedimage** in `info_image()` 174

**usehostfonts** in `set_option()` 28

**usehypertextencoding** in `set_option()` 28  
**uselayers** in `open_pdi_document` 189  
**usematchbox** in `create_annotation()` 218  
**usematchboxes** suboption for `wrap` in `fit_textflow()` 108  
**usercoordinates**  
 in `begin_item()` and the tag option 262  
 in `create_annotation()` 218  
 in `create_field/group()` 225  
 in `set_option()` 28  
**userlog** in `set_option()` 28  
**userpassword** in `begin_document()` 47  
**userunit**  
 in `begin/end_page_ext()` 55  
 suboption for `createrichtext` in `create_annotation()` 214  
**usestransparency** in `begin_document()` 46  
**usetags**  
 in `open_pdi_document` 189  
 in `open_pdi_page` 193

## V

**value**  
 in `poca_insert()` 40  
 suboption for `custom` in `create_annotation()` 214  
 suboption for `fieldlist` in `add_portfolio_file/folder()` 235  
 suboption for `properties` in `begin_mc()` and `mc_point()` 264  
**values** in `poca_insert()` 40  
**variantname** in `set_layer_dependency()` 60  
**version** sub-suboption for `portfolio` in `PDF_add_portfolio_file/folder()` 237  
**vertboxgap** keyword in `info_table()` 120  
**vertical**  
 in `info_font()` 73  
 in `load_font()` 68  
**verticalalign** in `fit_textflow()` 107  
**vertshrinking** keyword in `info_table()` 120  
**vertshrinklimit** in `fit_table()` 119  
**view** suboption for the `activate` suboption of `richmedia` in `create_annotation()` 251  
**viewarea** suboption for `viewerpreferences` in `begin/end_document()` 50  
**viewclip** suboption for `viewerpreferences` in `begin/end_document()` 50  
**viewerpreferences** in `begin_document()` and `end_document()` 45  
**viewports** in `begin/end_page_ext()` 55  
**views**  
 in `load_3d()` 242  
 suboption of `richmedia` in `create_annotation()` 250  
**visiblelayers** in `set_layer_dependency()` 60

## W

**weight**  
 in `begin_font()` 84  
 in `info_font()` 73  
**wellformed** keyword in `info_textline()` 94  
**width**  
 in `add_path_point()` 153  
 in `begin_glyph_ext()` 85  
 in `begin/end_page_ext()` 55  
 in `load_image()` 171  
 keyword in `info_*`() 130  
 keyword in `info_matchbox()` 134  
**Width** in `begin_item()` and the tag option 262  
**widthonly** in `begin_font()` 84  
**willembd** in `info_font()` 74  
**willsubset** in `info_font()` 74  
**windowposition** in `create_annotation()` 218  
**windowscale** in `create_annotation()` 218  
**wkt** suboption for the `coords` and `displaycoords` suboptions of `georeference` 239  
**wordspacing** in `set_text_option()`, `fit/info_textline()`, `fill_textblock()` and `add/create_textflow()` 78  
**worldpoints** suboption for `georeference` 239  
**wrap** in `fit_textflow()` 107  
**writingdirx**, **writingdiry** keywords in `info_textline()` 94

## X

**x1, y1, ..., x4, y4**  
 keywords in `info_*`() 130  
 keywords in `info_matchbox()` 134  
 keywords in `info_textflow()` 111  
**xadvancelist** in `fit/info_textline()` 91  
**xheight**  
 in `info_font()` 74  
 in `load_font()` 68  
 keyword in `info_textline()` 94  
**xid**  
 keyword in `info_graphics()` 180  
 keyword in `info_image()` 174  
 keyword in `info_pdi_page()` 196  
 suboption for `pdfvt` in `begin_template_ext()` 186  
**xrotate**  
 in `add_path_point()` 153  
 in `elliptical_arc()` 148  
**xstep** in `begin_pattern_ext()` 163  
**xsymheight**, **xsymwidth** suboption for the `barcode` option in `create_field/group()` 225  
**xvertline** keyword in `info_table()` 120

## Y

**yhorline** keyword in `info_table()` 120  
**yposition** suboption for `leader` in `fit/info_textline()` and `add/create_textflow()` 92

**ystep** in *begin\_pattern\_ext()* 163

## **Z**

### **zoom**

in *add\_nameddest()* and suboption for destination in *create\_action()*,

*create\_annotation()*, *create\_bookmark()* and *begin/end\_document()* 232  
in *create\_annotation()* 218  
in *define\_layer()* 58

# C Revision History

<b>Date</b>	<b>Changes</b>
December 16, 2014	▶ Updates for PDFlib 9.0.4
May 12, 2014	▶ Updates for PDFlib 9.0.3
December 17, 2013	▶ Updates for PDFlib 9.0.2
July 24, 2013	▶ Updates for PDFlib 9.0.1
March 11, 2013	▶ Updates for PDFlib 9.0.0
May 30, 2011	▶ Updates for PDFlib 8 VT Edition (internally 8.1.0)
May 30, 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

<b>Date</b>	<b>Changes</b>
<i>December 22, 2000</i>	▶ <i>ColdFusion documentation and additions for PDFlib 3.03; separate COM edition of the manual</i>
<i>August 8, 2000</i>	▶ <i>Delphi documentation and minor additions for PDFlib 3.02</i>
<i>July 1, 2000</i>	▶ <i>Additions and clarifications for PDFlib 3.01</i>
<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 options and keywords are listed separately in Appendix B, page 271.

## A

*abbreviated tagging* 262  
*action lists in option lists* 12  
*alignment (position option)* 127  
*All spot color name* 159  
*Author field* 256

## B

*Bézier curve* 145  
*Boolean values in option lists* 11

## C

*circles in option lists* 16  
*CMYK color* 13  
*cmyk keyword* 14  
*color functions* 157  
*color in option lists* 14  
*Creator field* 256  
*curves in option lists* 16

## D

*document and page functions* 41  
*document information fields* 255  
*document scope* 17  
*Dublin Core* 255

## F

*fast Web view* 43  
*Flash* 247  
*float and integer values in option lists* 12  
*floats in option lists* 11  
*font scope* 17  
*fontsize in option lists* 12  
*function scopes* 17

## G

*global options* 25  
*glyph scope* 17  
*graphics functions* 135, 175  
*gray keyword* 14

## H

*handles in option lists* 12

## I

*ICC Profiles* 160  
*ICC-based color* 13  
*iccbased keyword* 15  
*iccbasedcmyk keyword* 15  
*iccbasedgray keyword* 15  
*iccbasedrgb keyword* 15  
*Ideographic Variation Sequences (IVS)* 67  
*image functions* 167  
*import functions for PDF (PDI)* 187  
*info fields* 255  
*inline option lists for Textflows* 102  
*inner cell box for table cells* 113  
*invisible text* 78  
*IVS* 67

## K

*Keywords field* 256  
*keywords in option lists* 11

## L

*Lab color* 13  
*lab keyword* 14  
*landscape mode* 53  
*linearized PDF* 43  
*lines in option lists* 15  
*list values in option lists* 8

## M

*metadata* 257

## N

*nested option lists* 8  
*None spot color name* 159  
*numbers in option lists* 11

## O

*object scope* 17  
*option list syntax* 7  
*outline text* 78

## P

*page scope* 17  
*page size formats* 52  
*path painting and clipping* 149  
*path scope* 17

*pattern color* 14  
*pattern keyword* 15  
*pattern scope* 17  
*pCOS functions* 187, 199  
*PDF import functions (PDI)* 187  
*PDF Object Creation API (POCA)* 37  
*PDF\_activate\_item()* 263  
*PDF\_add\_nameddest()* 231  
*PDF\_add\_path\_point()* 151  
*PDF\_add\_portfolio\_folder()* 233, 234  
*PDF\_add\_table\_cell()* 112  
*PDF\_add\_textflow()* 95  
*PDF\_align()* 143  
*PDF\_arc()* 146, 147  
*PDF\_arcn()* 146  
*PDF\_begin\_document()* 41  
*PDF\_begin\_dpart()* 266  
*PDF\_begin\_font()* 84  
*PDF\_begin\_glyph\_ext()* 85  
*PDF\_begin\_item()* 258  
*PDF\_begin\_layer()* 61  
*PDF\_begin\_mc()* 264  
*PDF\_begin\_page\_ext()* 52  
*PDF\_begin\_pattern\_ext* 162  
*PDF\_begin\_template\_ext()* 181  
*PDF\_circle()* 146  
*PDF\_clip()* 150  
*PDF\_close\_font()* 68  
*PDF\_close\_graphics()* 178  
*PDF\_close\_image()* 171  
*PDF\_close\_pdi\_document()* 190  
*PDF\_close\_pdi\_page()* 193  
*PDF\_closepath\_fill\_stroke()* 150  
*PDF\_closepath\_stroke()* 149  
*PDF\_closepath()* 148  
*PDF\_concat()* 143  
*PDF\_continue\_text()* 82  
*PDF\_continue\_text2()* 82  
*PDF\_convert\_to\_unicode()* 23  
*PDF\_create\_3dview()* 242  
*PDF\_create\_action()* 226  
*PDF\_create\_annotation()* 211  
*PDF\_create\_bookmark()* 209  
*PDF\_create\_field()* 219  
*PDF\_create\_fieldgroup()* 220  
*PDF\_create\_gstate()* 139  
*PDF\_create\_pvf()* 34  
*PDF\_create\_textflow()* 101  
*PDF\_curveto()* 145  
*PDF\_define\_layer()* 57  
*PDF\_delete\_dl()* 33  
*PDF\_delete\_path()* 156  
*PDF\_delete\_pvf()* 35  
*PDF\_delete\_table()* 121  
*PDF\_delete\_textflow()* 111  
*PDF\_delete()* 33  
*PDF\_draw\_path()* 153  
*PDF\_ellipse()* 147  
*PDF\_elliptical\_arc()* 147  
*PDF\_encoding\_set\_char()* 87  
*PDF\_end\_document()* 42  
*PDF\_end\_dpart()* 266  
*PDF\_end\_font()* 85  
*PDF\_end\_glyph()* 86  
*PDF\_end\_item()* 262  
*PDF\_end\_layer()* 61  
*PDF\_end\_mc()* 264  
*PDF\_end\_pattern()* 164  
*PDF\_end\_template\_ext()* 182  
*PDF\_endpath()* 150  
*PDF\_fill\_graphicsblock()* 208  
*PDF\_fill\_imageblock()* 206  
*PDF\_fill\_pdfblock()* 207  
*PDF\_fill\_stroke()* 149  
*PDF\_fill\_textblock()* 204  
*PDF\_fill()* 149  
*PDF\_fit\_graphics()* 178  
*PDF\_fit\_image()* 172  
*PDF\_fit\_pdi\_page()* 194  
*PDF\_fit\_table()* 115  
*PDF\_fit\_textflow()* 103  
*PDF\_fit\_textline()* 89  
*PDF\_get\_apiname()* 22  
*PDF\_get\_buffer()* 51  
*PDF\_get\_errmsg()* 21  
*PDF\_get\_errnum()* 21  
*PDF\_get\_opaque()* 22  
*PDF\_get\_option()* 28  
*PDF\_get\_string()* 30  
*PDF\_get\_info\_font()* 69  
*PDF\_info\_graphics()* 179  
*PDF\_info\_image()* 172  
*PDF\_info\_matchbox()* 133  
*PDF\_info\_path()* 155  
*PDF\_info\_pdi\_page()* 195  
*PDF\_info\_pvf()* 35  
*PDF\_info\_table()* 120  
*PDF\_info\_textflow()* 109  
*PDF\_info\_textline()* 93  
*PDF\_lineto()* 145  
*PDF\_load\_3ddata()* 241  
*PDF\_load\_asset()* 247  
*PDF\_load\_font()* 63  
*PDF\_load\_graphics()* 175  
*PDF\_load\_iccprofile()* 160  
*PDF\_load\_image()* 167  
*PDF\_makespotcolor()* 159  
*PDF\_mc\_point()* 265  
*PDF\_moveto()* 145  
*PDF\_new\_dl()* 32  
*PDF\_new()* 32  
*PDF\_newz()* 32  
*PDF\_open\_pdi\_callback()* 189  
*PDF\_open\_pdi\_document()* 187  
*PDF\_open\_pdi\_page()* 191  
*PDF\_pcos\_get\_number()* 199

- PDF\_pcos\_get\_stream()* 200
- PDF\_pcos\_get\_string()* 199
- PDF\_poca\_delete()* 38
- PDF\_poca\_insert()* 39
- PDF\_poca\_new()* 37
- PDF\_poca\_remove()* 40
- PDF\_process\_pdi()* 197
- PDF\_rect()* 148
- PDF\_restore()* 139
- PDF\_resume\_page()* 56
- PDF\_rotate()* 142
- PDF\_save()* 138
- PDF\_scale()* 142
- PDF\_set\_graphics\_option()* 137
- PDF\_set\_gstate()* 139
- PDF\_set\_info()* 255
- PDF\_set\_info2()* 255
- PDF\_set\_layer\_dependency()* 58
- PDF\_set\_option()* 25
- PDF\_set\_text\_option()* 79
- PDF\_set\_text\_pos()* 80
- PDF\_setcolor()* 157
- PDF\_setfont()* 80
- PDF\_setlinewidth()* 138
- PDF\_setmatrix()* 144
- PDF\_shading\_pattern()* 164
- PDF\_shading()* 165
- PDF\_shfill()* 164
- PDF\_show\_xy()* 81
- PDF\_show\_xy2()* 81
- PDF\_show()* 81
- PDF\_show2()* 81
- PDF\_skew()* 143
- PDF\_stringwidth()* 82
- PDF\_stringwidth2()* 82
- PDF\_stroke()* 149
- PDF\_suspend\_page()* 55
- PDF\_translate()* 142
- PDF/A or PDF/X output intent* 197
- PDFlib Personalization Server (PPS)* 203
- PDI (PDF import)* 187
- POCA (PDF Object Creation API)* 37
- polylines in option lists* 15
- PPS (PDFlib Personalization Server)* 203

## R

- raster image functions* 167
- rectangles in option lists* 16
- RGB color* 13
- rgb keyword* 14
- rich media* 247

## S

- scopes* 17
- separation color space* 13
- setup functions* 32
- skewing* 143

- spot color (separation color space)* 13
- spot keyword* 14
- spotname keyword* 15
- standard page sizes* 52
- strings in option lists* 10
- Subject field* 255
- subscript* 78
- superscript* 78
- SVG* 175
- syntax of option lists* 7

## T

- table formatting* 112
- template scope* 17
- text appearance options* 89
- text filter options* 89
- text functions* 63
- Textflow: inline option lists* 102
- Title field* 255
- Trapped field* 256

## U

- Unichar values in option lists* 11
- Unicode ranges in option lists* 11
- Unquoted string values in option lists* 9

## V

- vector graphics functions* 175

## W

- web-optimized PDF* 43

## X

- XMP metadata* 257

**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 [groups.yahoo.com/neo/groups/pdflib/info](http://groups.yahoo.com/neo/groups/pdflib/info)

**Licensing contact**

[sales@pdflib.com](mailto:sales@pdflib.com)

**Support**

[support@pdflib.com](mailto:support@pdflib.com) (*please include your license number*)

