

PDF plugin

Create and Render PDFs with Hollywood

Andreas Falkenhahn

Table of Contents

1	General information	1
1.1	Introduction	1
1.2	Terms and conditions	1
1.3	Requirements	2
1.4	Installation	2
2	About the plugin	5
2.1	The two flavours	5
2.2	Credits	5
2.3	Frequently asked questions	5
2.4	Future	6
2.5	History	6
3	Viewing PDFs	7
3.1	Overview	7
3.2	Loading pages as vector brushes	7
3.3	Loading PDFs as anims	8
4	Creating PDFs	9
4.1	Coordinate system	9
4.2	Graphics mode	9
4.3	Painting paths	10
4.4	Painting text	11
4.5	Colors	12
4.6	Font types	12
4.7	Base14 fonts	13
4.8	Type1 fonts	13
4.9	TrueType fonts	13
4.10	CID fonts	14
4.11	Encodings	14
5	Tutorial	19
5.1	Tutorial	19
6	General functions	21
6.1	pdf.CloseDocument	21
6.2	pdf.CreateDocument	21
6.3	pdf.DeviceToPage	21
6.4	pdf.FindNext	23
6.5	pdf.FindPrev	23
6.6	pdf.FindStart	24

6.7	pdf.FreePage	25
6.8	pdf.GetBookmarks	25
6.9	pdf.GetBoundedText	26
6.10	pdf.GetBrush	27
6.11	pdf.GetBrushFromPage	28
6.12	pdf.GetCharBox	29
6.13	pdf.GetCharIndexAtPos	30
6.14	pdf.GetCharOrigin	30
6.15	pdf.GetCropBox	31
6.16	pdf.GetFindResult	31
6.17	pdf.GetLastError	32
6.18	pdf.GetMediaBox	33
6.19	pdf.GetMetaText	33
6.20	pdf.GetObjectType	34
6.21	pdf.GetPageLabel	35
6.22	pdf.GetPageLen	35
6.23	pdf.GetPageLinks	36
6.24	pdf.GetRects	37
6.25	pdf.GetText	38
6.26	pdf.GetVersion	38
6.27	pdf.IsPDF	39
6.28	pdf.LoadPage	39
6.29	pdf.OpenDocument	40
6.30	pdf.PageToDevice	40
7	Annotation methods	43
7.1	annot:SetBorderStyle	43
7.2	annot:SetCMYKColor	44
7.3	annot:SetFreeTextAnnot2PointCalloutLine	44
7.4	annot:SetFreeTextAnnot3PointCalloutLine	45
7.5	annot:SetFreeTextAnnotDefaultStyle	45
7.6	annot:SetFreeTextAnnotLineEndingStyle	46
7.7	annot:SetGrayColor	46
7.8	annot:SetLineAnnotCaption	47
7.9	annot:SetLineAnnotLeader	47
7.10	annot:SetLineAnnotPosition	48
7.11	annot:SetLinkAnnotBorderStyle	49
7.12	annot:SetLinkAnnotHighlightMode	49
7.13	annot:SetMarkupAnnotCloudEffect	50
7.14	annot:SetMarkupAnnotCreationDate	50
7.15	annot:SetMarkupAnnotIntent	51
7.16	annot:SetMarkupAnnotInteriorCMYKColor	52
7.17	annot:SetMarkupAnnotInteriorGrayColor	52
7.18	annot:SetMarkupAnnotInteriorRGBColor	53
7.19	annot:SetMarkupAnnotInteriorTransparent	53
7.20	annot:SetMarkupAnnotPopup	54

7.21	annot:SetMarkupAnnotQuadPoints	54
7.22	annot:SetMarkupAnnotRectDiff	55
7.23	annot:SetMarkupAnnotSubject	55
7.24	annot:SetMarkupAnnotTitle	55
7.25	annot:SetMarkupAnnotTransparency	56
7.26	annot:SetNoColor	56
7.27	annot:SetPopupAnnotOpened	57
7.28	annot:SetRGBColor	57
7.29	annot:SetTextAnnotIcon	58
7.30	annot:SetTextAnnotOpened	58
8	Destination methods	61
8.1	dest:SetFit	61
8.2	dest:SetFitB	61
8.3	dest:SetFitBH	61
8.4	dest:SetFitBV	62
8.5	dest:SetFitH	62
8.6	dest:SetFitR	63
8.7	dest:SetFitV	64
8.8	dest:SetXYZ	64
9	Document methods	65
9.1	doc:AddPage	65
9.2	doc:AddPageLabel	65
9.3	doc:AttachFile	66
9.4	doc:CreateExtGState	67
9.5	doc:CreateImageFromBrush	67
9.6	doc:CreateImageFromMem	68
9.7	doc:CreateOutline	69
9.8	doc:Free	69
9.9	doc:GetCurrentEncoder	70
9.10	doc:GetCurrentPage	70
9.11	doc:GetEncoder	70
9.12	doc:GetError	71
9.13	doc:GetErrorDetail	71
9.14	doc:GetFont	72
9.15	doc:GetInfoAttr	72
9.16	doc:GetPageByIndex	73
9.17	doc:GetPageLayout	73
9.18	doc:GetPageMode	74
9.19	doc:GetViewerPreference	74
9.20	doc:InsertPage	75
9.21	doc:LoadFont	75
9.22	doc:LoadJPEGImage	76
9.23	doc:LoadPNGImage	77
9.24	doc:LoadRawImage	78

9.25	doc:LoadTTFont	79
9.26	doc:LoadType1Font	80
9.27	doc:ResetError	80
9.28	doc:SaveToFile	81
9.29	doc:SetCompressionMode	81
9.30	doc:SetCurrentEncoder	82
9.31	doc:SetEncryptionMode	83
9.32	doc:SetInfoAttr	83
9.33	doc:SetInfoDateAttr	84
9.34	doc:SetOpenAction	85
9.35	doc:SetPageLayout	85
9.36	doc:SetPageMode	86
9.37	doc:SetPagesConfiguration	87
9.38	doc:SetPassword	88
9.39	doc:SetPermission	88
9.40	doc:SetViewerPreference	89
9.41	doc:UseCNSEncodings	90
9.42	doc:UseCNSFonts	91
9.43	doc:UseCNTEncodings	91
9.44	doc:UseCNTFonts	92
9.45	doc:UseJPEncodings	93
9.46	doc:UseJPFonts	93
9.47	doc:UseKREncodings	94
9.48	doc:UseKRFonts	95
9.49	doc:UseUTFEncodings	96
10	Encoder methods	97
10.1	encoder:GetByteType	97
10.2	encoder:GetType	97
10.3	encoder:GetUnicode	97
10.4	encoder:GetWritingMode	98
11	ExtGState methods	99
11.1	extgs:SetAlphaFill	99
11.2	extgs:SetAlphaStroke	99
11.3	extgs:SetBlendMode	100
12	Font methods	101
12.1	font:GetAscent	101
12.2	font:GetBBox	101
12.3	font:GetCapHeight	101
12.4	font:GetDescent	102
12.5	font:GetEncodingName	102
12.6	font:GetFontName	102
12.7	font:GetUnicodeWidth	103

12.8	font:GetXHeight	103
12.9	font:MeasureText	104
12.10	font:TextWidth	104
13	Image methods	107
13.1	image:AddSMask	107
13.2	image:GetBitsPerComponent	107
13.3	image:GetColorSpace()	107
13.4	image:GetHeight	108
13.5	image:GetSize	108
13.6	image:GetWidth	108
13.7	image:SetColorMask	109
13.8	image:SetMaskImage	109
14	Outline methods	111
14.1	outline:SetDestination	111
14.2	outline:SetOpened	111
15	Page methods	113
15.1	page:Arc	113
15.2	page:BeginText	113
15.3	page:Circle	114
15.4	page:Clip	114
15.5	page:ClosePath	115
15.6	page:ClosePathEofillStroke	115
15.7	page:ClosePathFillStroke	115
15.8	page:ClosePathStroke	116
15.9	page:Concat	116
15.10	page:CreateCircleAnnot	117
15.11	page:CreateDestination	118
15.12	page:CreateFreeTextAnnot	118
15.13	page:CreateHighlightAnnot	119
15.14	page:CreateLineAnnot	119
15.15	page:CreateLinkAnnot	120
15.16	page:CreatePopupAnnot	120
15.17	page:CreateProjectionAnnot	121
15.18	page:CreateSquareAnnot	121
15.19	page:CreateSquigglyAnnot	122
15.20	page:CreateStampAnnot	122
15.21	page:CreateStrikeOutAnnot	123
15.22	page:CreateTextAnnot	124
15.23	page:CreateTextMarkupAnnot	125
15.24	page:CreateUnderlineAnnot	126
15.25	page:CreateURILinkAnnot	126
15.26	page:CreateWidgetAnnot	127

15.27	page:CurveTo	127
15.28	page:CurveTo2	128
15.29	page:CurveTo3	128
15.30	page:DrawImage	129
15.31	page:Ellipse	129
15.32	page:EndPath	130
15.33	page:EndText	130
15.34	page:EoClip	130
15.35	page:Eofill	131
15.36	page:EofillStroke	131
15.37	page:ExecuteXObject	131
15.38	page:Fill	132
15.39	page:FillStroke	132
15.40	page:GetCharSpace	132
15.41	page:GetCMYKFill	133
15.42	page:GetCMYKStroke	133
15.43	page:GetCurrentFont	134
15.44	page:GetCurrentFontSize	134
15.45	page:GetCurrentPos	135
15.46	page:GetCurrentTextPos	135
15.47	page:GetDash	135
15.48	page:GetFillingColorSpace	136
15.49	page:GetFlat	136
15.50	page:GetGMode	137
15.51	page:GetGrayFill	137
15.52	page:GetGrayStroke	137
15.53	page:GetGStateDepth	138
15.54	page:GetHeight	138
15.55	page:GetHorizontalScaling	139
15.56	page:GetLineCap	139
15.57	page:GetLineJoin	139
15.58	page:GetLineWidth	140
15.59	page:GetMiterLimit	140
15.60	page:GetRGBFill	140
15.61	page:GetRGBStroke	141
15.62	page:GetStrokingColorSpace	141
15.63	page:GetTextLeading	142
15.64	page:GetTextMatrix	142
15.65	page:GetTextRenderingMode	143
15.66	page:GetTextRise	143
15.67	page:GetTransMatrix	143
15.68	page:GetWidth	144
15.69	page:GetWordSpace	144
15.70	page:GRestore	145
15.71	page:GSave	145
15.72	page:LineTo	146
15.73	page:MeasureText	146

15.74	page:MoveTextPos	147
15.75	page:MoveTo	147
15.76	page:MoveToNextLine	148
15.77	page:Rectangle	148
15.78	page:SetCharSpace	149
15.79	page:SetCMYKFill	149
15.80	page:SetCMYKStroke	149
15.81	page:SetDash	150
15.82	page:SetExtGState	150
15.83	page:SetFlat	151
15.84	page:SetFontAndSize	151
15.85	page:SetGrayFill	151
15.86	page:SetGrayStroke	152
15.87	page:SetHeight	152
15.88	page:SetHorizontalScaling	153
15.89	page:SetLineCap	153
15.90	page:SetLineJoin	153
15.91	page:SetLineWidth	154
15.92	page:SetMiterLimit	154
15.93	page:SetRGBFill	155
15.94	page:SetRGBStroke	155
15.95	page:SetRotate	156
15.96	page:SetSize	156
15.97	page:SetSlideShow	157
15.98	page:SetTextLeading	158
15.99	page:SetTextMatrix	159
15.100	page:SetTextRenderingMode	159
15.101	page:SetTextRise	160
15.102	page:SetWidth	160
15.103	page:SetWordSpace	161
15.104	page:SetZoom	161
15.105	page:ShowText	161
15.106	page:ShowTextNextLine	162
15.107	page:Stroke	162
15.108	page:TextOut	162
15.109	page:TextRect	163
15.110	page:TextWidth	164
Appendix A Licenses		165
A.1	LibHaru license	165
A.2	LuaHPDF license	165
A.3	PDFium license	165
Index		167

1 General information

1.1 Introduction

This is a plugin for Hollywood that allows you to easily create PDF documents from Hollywood scripts. On top of that, the PDF plugin can also open existing PDF documents and convert their pages into Hollywood brushes. In fact, when converting PDF pages into Hollywood brushes, the plugin will create vector brushes for you which can be scaled, rotated and transformed without any losses in quality (unless bitmap graphics are embedded inside the PDF document of course).

The plugin comes with over 200 functions for creating PDF documents of all sorts. It supports graphics primitives, text in different encodings including Unicode, embedding fonts as well as images and Hollywood brushes inside PDF documents. On top of that the plugin supports the creation of password-protected PDF documents, encrypted PDF documents, compression, file attachments, annotations, extended graphics states, info dictionaries, RGB, CMYK and gray color spaces, different viewing modes, transition effects, links, and permission flags for PDF documents. Transformation of PDF objects is fully supported too. Finally, the plugin can also create PDF documents with an easy-to-navigate outline that can be used as a table of contents as well.

The plugin also has support for extracting text from PDF pages, getting all bookmarks in a document, handling links on PDF pages, and it is also possible to search pages. Furthermore, the plugin allows you to query the position of text on PDF pages, making it possible to implement text marking functionality, for instance.

The plugin comes with extensive documentation in various formats like PDF (of course), HTML, AmigaGuide, and CHM that contains detailed descriptions about all functions and methods offered by the plugin. On top of that, over 25 example scripts are included in the distribution archive to get you started really quickly.

All of this makes this plugin the ultimate PDF tool for Hollywood that contains everything you need to empower your app with PDF support!

1.2 Terms and conditions

The pdf.hwp plugin is © Copyright 2013-2026 by Andreas Falkenhahn (in the following referred to as "the author"). All rights reserved.

The program is provided "as-is" and the author cannot be made responsible of any possible harm done by it. You are using this program absolutely at your own risk. No warranties are implied or given by the author.

This plugin may be freely distributed as long as the following three conditions are met:

1. No modifications must be made to the plugin.
2. It is not allowed to sell this plugin.
3. If you want to put this plugin on a coverdisc, you need to ask for permission first.

This software uses LibHaru by Takeshi Kanno and Antony Dovgal. See [Section A.1 \[LibHaru license\]](#), [page 165](#), for details.

This software uses LuaHPDF by Kurt Jung. See [Section A.2 \[LuaHPDF license\]](#), page 165, for details.

This software uses PDFium by the PDFium authors. See [Section A.3 \[PDFium license\]](#), page 165, for details.

All trademarks are the property of their respective owners.

DISCLAIMER: THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDER AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.3 Requirements

- Hollywood 7.1 or better
- on macOS, the plugin requires at least 10.9 on x86 and x64 systems and 10.5 on PowerPC systems
- on Android, at least version 5.0 is required
- if you use WinUAE, you need at least WinUAE 4.2.1 or the plugin can crash because of a bug in WinUAE's 68020 emulation

1.4 Installation

Installing pdf.hwp is straightforward and simple: Just copy the files `pdf.hwp` and `pdf-lite.hwp` for the platform of your choice to Hollywood's plugins directory. The location where Hollywood plugins must be installed differs from platform to platform. Here is a breakdown of the different search paths that Hollywood will scan for plugins:

- AmigaOS and compatibles: Plugins can either be stored in a directory named `Plugins` that is in the same directory as the Hollywood interpreter or they can be installed globally in `LIBS:Hollywood`. Hollywood will first look for plugins in the `Plugins` directory in its program directory, then it will scan `LIBS:Hollywood`.

- Android: The Android version of Hollywood also supports Hollywood plugins. You have to copy them to the directory `Hollywood/Plugins` on your SD card. Hollywood will scan this location on every startup and load all plugins from there.
- Linux: Plugins must be stored in a directory named `Plugins` (case sensitive!) that is in the same directory as the Hollywood interpreter.
- macOS: `~/Library/Application Support/AirsoftSoftwair/Hollywood/Plugins` is the directory that should be used for storing plugins. If it doesn't exist, just create it. Alternatively, you can also store plugins inside Hollywood's app bundle in `Hollywood.app/Contents/Resources/Plugins` but this isn't recommended because it requires you to modify the Hollywood distribution and it might also require administrator privileges.
- Windows: `C:\Users\...\AppData\Local\AirsoftSoftwair\Hollywood\Plugins` is the directory that should be used for storing plugins. If it doesn't exist, just create it. Alternatively, you can also store plugins in the `Plugins` directory in Hollywood's installation directory, e.g. in `C:\Program Files\Hollywood\Plugins` but this isn't recommended because it requires you to modify the Hollywood distribution and it might also require administrator privileges to write to Hollywood's installation directory.

On Windows you should also copy the file `pdf.chm` to the `Docs` directory which you can find in the same directory as the `Plugins` directory (see above). After you have copied `pdf.chm` to the `Docs` directory, you will be able to get online help by pressing F1 when the cursor is over a `pdf.hwp` function in the Hollywood IDE.

On macOS copy the `pdf` directory that is inside the `Docs` directory of `pdf.hwp`'s distribution archive to the `Docs` directory which you can find in the same directory as the `Plugins` directory (see above).

On Linux copy the `pdf` directory that is inside the `Docs` directory of `pdf.hwp`'s distribution archive to the `Docs` directory of your Hollywood installation.

2 About the plugin

2.1 The two flavours

The PDF plugin comes in two different flavours: pdf.hwp and pdf-lite.hwp. The difference between the two is that pdf-lite.hwp doesn't contain the PDF renderer which means that it is much smaller than pdf.hwp which contains a fully-featured PDF renderer. So if you only need to create PDFs, you can just use the pdf-lite flavour of the plugin and it's not necessary to load the full version of the plugin.

Note that functions like `pdf.OpenDocument()` are also unavailable in the pdf-lite flavour so it's also not possible to parse PDFs with the light version of the plugin. The only thing the pdf-lite flavour can do is creating PDFs using `pdf.CreateDocument()` and its corresponding methods.

2.2 Credits

The pdf.hwp plugin was written by Andreas Falkenhahn, based on work done by Takeshi Kanno, Antony Dovgal, Kurt Jung and the PDFium authors. Special thanks go to Sebastian Bauer for adding rudimentary wide character support to clib2 so that PDFium can be compiled on AmigaOS 4 as well. Further thanks go to Stefan "Bebbo" Franke for maintaining a recent version of gcc that can compile for the Motorola 680x0 series.

If you need to contact me, you can either send an e-mail to andreas@airsoftsoftware.de or use the contact form on <http://www.hollywood-mal.com>.

2.3 Frequently asked questions

This section covers some frequently asked questions. Please read them first before asking on the forum because your problem might have been covered here.

Q: How can I modify existing PDF documents?

A: That's currently not supported but planned for a future version of the plugin.

Q: Why doesn't the plugin support the conversion of PDF pages to vector brushes on AROS?

A: That's because PDFium requires a compiler capable of handling C++11 and wide characters which is currently unavailable for AROS. But this will hopefully change in the future so that AROS users can convert PDF pages into Hollywood brushes too.

Q: Why aren't Chinese/Japanese/Korean (CJK) characters drawn correctly in my document?

A: Make sure you have a TrueType font that has CJK support installed. For example, install Konatu on your system and CJK characters should be drawn correctly.

Q: The 68k version of the plugin doesn't work under OS4 emulation.

A: It seems that the OS4 JIT has problems dealing with the plugin' PDF renderer which is a 5 MB binary. If you really want to use the 68k version of the plugin on OS4, you need to disable JIT for the file LIBS:Hollywood/pdf.hwp. Then it should work.

Q: Is there a Hollywood forum where I can get in touch with other users?

A: Yes, please check out the "Community" section of the official Hollywood Portal online at <http://www.hollywood-mal.com>.

Q: Where can I ask for help?

A: There's an active forum at <http://forums.hollywood-mal.com>. You're welcome to join it and ask your question there.

Q: I have found a bug.

A: Please post about it in the "Bugs" section of the forum.

2.4 Future

Here are some things that are on my to do list:

- add support for rendering PDF pages on AROS
- add support for editing existing PDF documents

Don't hesitate to contact me if the plugin lacks a certain feature that is important for your project.

2.5 History

Please see the file `history.txt` for a complete change log of the plugin.

3 Viewing PDFs

3.1 Overview

There are two different methods of viewing PDF documents with the plugin in Hollywood: You can either load individual PDF pages as vector brushes or you can load an entire PDF document as a Hollywood animation in which the document's pages are simply mapped to individual anim frames. Whatever way you choose, the plugin will always map PDF pages to vector objects in Hollywood so that they can be scaled, rotated, and transformed without any losses in quality.

To use the plugin from your Hollywood script, you first need to initialize the plugin at the beginning of your script by using the following line:

```
@REQUIRE "pdf"
```

There are also some additional arguments that you can pass to the `@REQUIRE` preprocessor command. The following arguments are currently available:

NoVectorAnim:

When loading PDFs as Hollywood anims, the plugin will automatically create vector animations if the Hollywood version used with the plugin supports it. If you don't want that, set this tag to **True**. Note that if you set this tag to **True**, there will be quality losses when scaling or transforming the animation that contains the PDF pages. Defaults to **False**. (V1.3)

3.2 Loading pages as vector brushes

The plugin can be used to conveniently load PDF pages as vector brushes. Just pass the desired page to load in the user tag named **Page** and the plugin will load the specified page. For example, this code loads page 10 as vector brush 1:

```
LoadBrush(1, "test.pdf", {UserTags = {Page = 10}})
```

If the **Page** user tag isn't specified, it will default to 1 which means that the first page will be loaded. By default, the page backgrounds will be filled with a static white background. If you want the background to be transparent instead, set the **Transparent** user tag to **True**, e.g. like this:

```
LoadBrush(1, "test.pdf", {UserTags = {Page = 10, Transparent = True}})
```

In that case, the page's background will be made transparent. Finally, you can also specify a user tag named **Password** to specify a password in case the PDF file is password-protected.

Note that in case you need to load multiple pages as vector brushes at the same time, it's more memory efficient to use `pdf.OpenDocument()` to open the PDF and then `pdf.GetBrush()` to get vector brushes of the individual pages. If you use `LoadBrush()` or `@BRUSH`, the whole PDF will be loaded into memory for each page you convert to a vector brush, which can be quite memory hungry. If you use `pdf.OpenDocument()` and then `pdf.GetBrush()` to get vector brushes of the individual pages then PDF will only be stored in memory once.

Here is an example how to convert PDF pages to vector brushes using the `pdf.OpenDocument()` function and the `pdf.GetBrush()` command:

```
pdf.OpenDocument(1, "test.pdf")
```

```
pdf.GetBrush(1, 1, 1)
DisplayBrush(1, #CENTER, #CENTER)
FreeBrush(1)
pdf.CloseDocument(1)
```

The code above will open the PDF document named `test.pdf` and convert its first page to a vector brush. It will then show this vector brush in the center of the display. Note that the vector brush will still depend on the PDF document so it is not allowed to call `pdf.CloseDocument()` on the document while you still need the brush. That's why we free the brush first and close the document afterwards. Otherwise there will be an error.

You can find out the number of pages in the PDF document by first getting the object type for PDF documents and then using Hollywood's `GetAttribute()` function, like so:

```
PDF_DOCUMENT = pdf.GetObjectType()
numpages = GetAttribute(PDF_DOCUMENT, 1, #PDFATTRPAGES)
```

The code above gets the number of pages from the PDF document that uses the identifier 1 and stores it in the variable `numpages`.

3.3 Loading PDFs as anims

Alternatively, the plugin offers to load an entire PDF document into a Hollywood anim object. You can then access the individual pages by simply obtaining the anim's frames.

Here's how to load a PDF document as a Hollywood anim:

```
LoadAnim(1, "test.pdf", {FromDisk = True})
For Local k = 1 To GetAttribute(#ANIM, 1, #ATTRNUMFRAMES)
    DisplayAnimFrame(1, #CENTER, #CENTER, k)
    WaitLeftMouse
Next
```

The code above shows all pages of a PDF document. You need to press the left mouse button to skip to the next page.

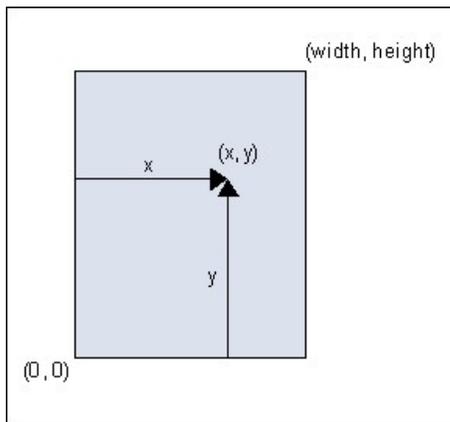
Note that we set `FromDisk` tag to `True` in our `LoadAnim()` call. This is very important because otherwise all PDF pages will be loaded and buffered in memory which can be a huge waste with larger PDF documents.

Of course, you could also load the PDF document with the `@ANIM` preprocessor command instead of `LoadAnim()`.

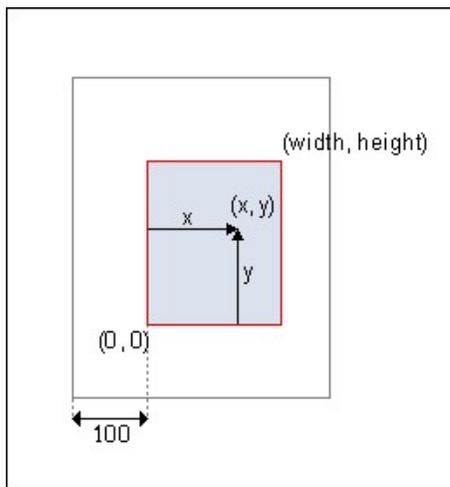
4 Creating PDFs

4.1 Coordinate system

Note that PDF documents use a different coordinate system than Hollywood. In the default coordinate system of PDF, shown below, the lower-left corner is at coordinates $(0, 0)$, and the upper-right corner is at coordinates $(\text{width}, \text{height})$. The default resolution is 72dpi. In Hollywood the upper-left corner is at $(0, 0)$.



An application can change the coordinate system by invoking `page:Concat()`. For example, if an application invokes `page:Concat(0.5, 0, 0, 0.5, 100, 100)` in the default state, the coordinate system shown above is transformed to the new system shown in the figure below:

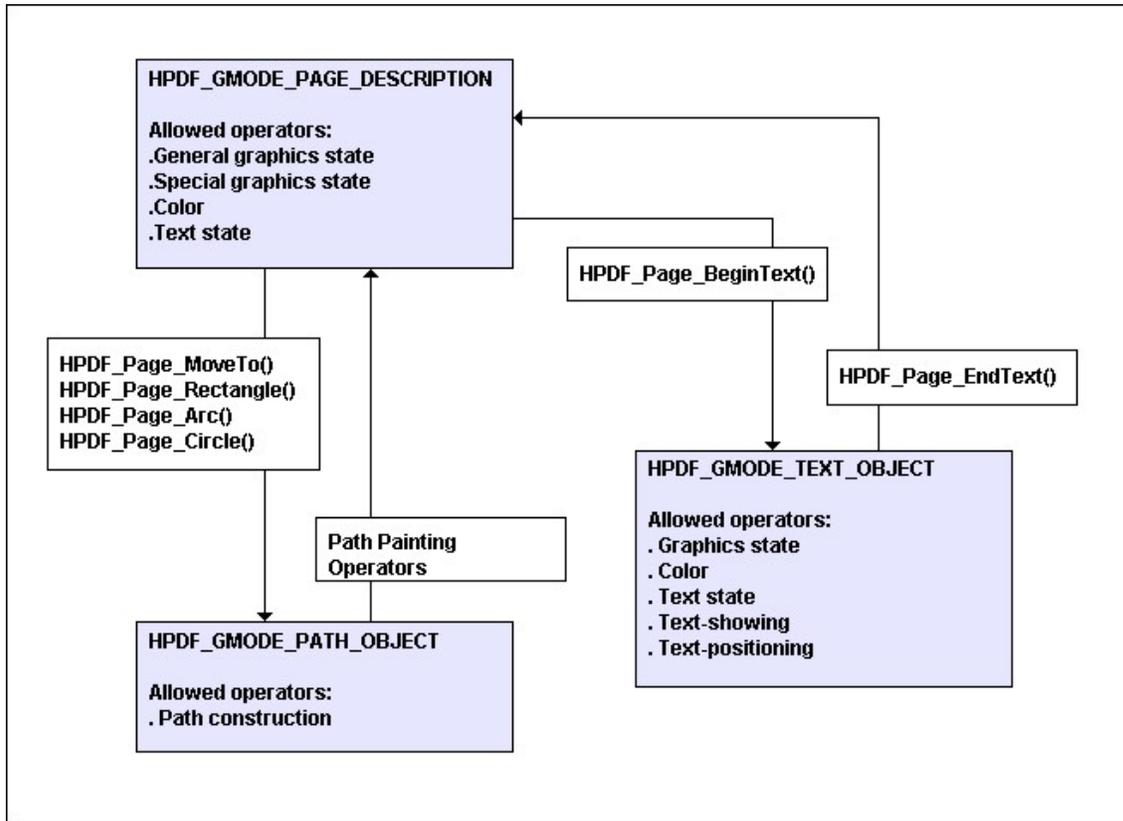


4.2 Graphics mode

In the plugin's API, each page object maintains a flag named "graphics mode". The graphics mode corresponds to the graphics object of the PDF specification.

The graphics mode is changed by invoking particular functions. The functions that can be invoked are decided by the value of the graphics mode.

The following figure shows the relationships of the graphics mode.



4.3 Painting paths

A path is composed of straight and curved line segments. Paths define shapes and regions. Vector graphics are drawn by the following steps:

1. Set graphics states (such as line width, dash pattern, color...) using graphics state operators or color operators.
2. Start new path using `page:MoveTo()`, `page:Rectangle()`, `page:Arc()`, or `page:Circle()`.
3. Append to path using path construction operators.
4. Stroke or paint the path using path painting operators.

Here is a list of graphics state operators:

```
page:Concat()
page:SetDash()
page:SetFlat()
page:SetLineCap()
page:SetLineJoin()
page:SetLineWidth()
page:SetMiterLimit()
```

Here is a list of color operators:

```
page:SetCMYKFill()
```

```

page:SetCMYKStroke()
page:SetGrayFill()
page:SetGrayStroke()
page:SetRGBFill()
page:SetRGBStroke()

```

Here is a list of path construction operators:

```

page:Arc()
page:Circle()
page:CurveTo()
page:CurveTo2()
page:CurveTo3()
page:LineTo()
page:MoveTo()
page:Rectangle()

```

Here is a list of path painting operators:

```

page:ClosePathFillStroke()
page:ClosePathEofillStroke()
page:ClosePathStroke()
page:Eofill()
page:EofillStroke()
page:EndPath()
page:Fill()
page:FillStroke()
page:Stroke()

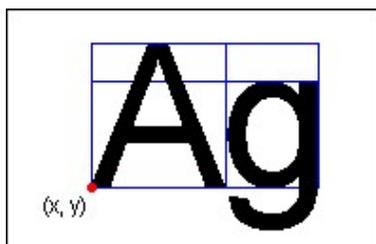
```

4.4 Painting text

Text is drawn by the following steps:

1. Start drawing text by invoking `page:BeginText()`.
2. Set text states (such as font, filling color...) using text state operators or color operators. At least `page:SetFontAndSize()` must be invoked once before invoking text painting operators.
3. Set text positioning by invoking text positioning operators.
4. Show text by invoking text painting operators.
5. Repeat steps 2 to 4 if necessary.
6. Finish drawing text by invoking `page:EndText()`.

The figure below explains text positioning:



You can see that, in contrast to Hollywood's coordinate system, the PDF document's coordinate system for placing text starts at the bottom and extends upwards.

Here is a list of text state operators:

```
page:SetCharSpace()
page:SetFontAndSize()
page:SetHorizontalScaling()
page:SetTextLeading()
page:SetTextRenderingMode()
page:SetTextRise()
page:SetWordSpace()
```

Here is a list of text positioning operators:

```
page:MoveTextPos()
page:SetTextMatrix()
```

Here is a list of text painting operators:

```
page:ShowText()
page:ShowTextNextLine()
page:TextOut()
page:TextRect()
```

4.5 Colors

Colors are specified using three real numbers (i.e. ones with a decimal point) in the form R G B where each number defines the amount of red (R), green (G) and blue (B) in a color. The valid numbers are from 0.0 to 1.0 inclusive.

4.6 Font types

There are several types of fonts available in the plugin.

Base14 font:

The built-in font of PDF. Available in all viewer applications.

Type1 font:

A font format used by PostScript.

TrueType font:

Widely used outline font format.

CID font: Font format for multi-byte characters. Developed by Adobe.

Hollywood scripts can use `doc:GetFont()` to get a font handle. Before that, one of the following functions must be used to load the font before invoking `doc:GetFont()`: (except for Base14 fonts, those are always available and needn't be loaded)

```
HPDF_LoadType1FontFromFile()
HPDF_LoadTTFontFromFile()
HPDF_LoadTTFontFromFile2()
HPDF_UseCNSFonts()
HPDF_UseCNTFonts()
HPDF_UseJPFonts()
HPDF_UseKRFonts()
```

4.7 Base14 fonts

Base14 fonts are built into PDF and all viewer applications can display these fonts. An application can get a Base14 font handle any time by invoking `doc:GetFont()`. PDF files which use base14 fonts are smaller than those which use other type of fonts. Moreover, PDF processing is faster because there is no need to load external fonts. However, Base14 fonts are only able to display the Latin-1 character set. To use other character sets, an application must use other fonts.

The following are built-in Base14 fonts. They are available in every PDF viewer:

```
Courier
Courier-Bold
Courier-Oblique
Courier-BoldOblique
Helvetica
Helvetica-Bold
Helvetica-Oblique
Helvetica-BoldOblique
Times-Roman
Times-Bold
Times-Italic
Times-BoldItalic
Symbol
ZapfDingbats
```

4.8 Type1 fonts

Type1 is a format of outline fonts developed by Adobe. An AFM file is necessary to use an external Type1 font with the plugin. When a Hollywood script uses an external Type1 font, it has to invoke `doc:LoadType1Font()` before invoking `doc:GetFont()`. The return value of `doc:LoadType1Font()` is used as the font name parameter of `doc:GetFont()`. If a PFA/PFB file is specified when invoking `doc:LoadType1Font()`, the glyph data of the font is embedded into the PDF file. Otherwise, only metrics data in AFM file is embedded.

Here is an example:

```
fontname = doc:LoadType1Font("a0100131.afm", "a0100131.pfb")
hfont = doc:GetFont(fontname, "CP1250")
page:SetFontAndSize(hfont, 10.5)
```

4.9 TrueType fonts

The PDF plugin can use TrueType fonts. There are two types of TrueType fonts: The first format, which uses the ".ttf" extension, contains only one font in its file. The second format, which uses the ".ttc" extension, contains multiple fonts in its file. That is why `doc:LoadTTFont()` has a parameter which is used to specify the index of the font to load. If the additional parameter `embedding` is set to `True` when invoking `doc:LoadTTFont()`, the subset of the font is embedded into the PDF file. If not, only the marix data is stored in the PDF file. In this case a viewer application may use an alternative font if it cannot find the font.

Here is an example:

```
fontname = doc:LoadTTFont("arial.ttf", True)
hfont = doc:GetFont(fontname, "CP1250")
page:SetFontAndSize(hfont, 10.5)
```

Note that the plugin can use only TrueType fonts which have a Unicode cmap and one of the following tables: "OS/2", "cmap", "cvt ", "fpgm", "glyf", "head", "hhea", "hmtx", "loca", "maxp", "name", "post", "prep".

4.10 CID fonts

CID fonts are a multi-byte character font format developed by Adobe. Two simplified Chinese fonts, one traditional Chinese fonts, four Japanese fonts, and four Korean fonts are available in the plugin. Hollywood scripts have to invoke the following functions once before using CID fonts:

`doc:UseCNSFonts()`

Makes simplified Chinese fonts (SimSun, SimHei) become available.

`doc:UseCNTFonts()`

Makes traditional Chinese fonts (MingLiU) become available.

`doc:UseJPFonts()`

Makes Japanese fonts (MS-Mincyo, MS-Gothic, MS-PMincyo, MS-PGothic) become available.

`doc:UseKRFonts()`

Makes Korean fonts (Batang, Dotum, BatangChe, DotumChe) become available.

Here is an example:

```
doc:UseJPFonts()
doc:UseJPEncodings()
hfont = doc:GetFont("MS-Mincyo", "90ms-RKSJ-H")
page:SetFontAndSize(hfont, 10.5)
```

4.11 Encodings

The following single-byte encodings are available in the plugin. Hollywood scripts can get an encoding handle by using `doc:GetEncoder()`:

`StandardEncoding`

The default encoding of PDF

`MacRomanEncoding`

The standard encoding of macOS

`WinAnsiEncoding`

The standard encoding of Windows

`FontSpecific`

Use the built-in encoding of a font

IS08859-2	Latin Alphabet No.2
IS08859-3	Latin Alphabet No.3
IS08859-4	Latin Alphabet No.4
IS08859-5	Latin Cyrillic Alphabet
IS08859-6	Latin Arabic Alphabet
IS08859-7	Latin Greek Alphabet
IS08859-8	Latin Hebrew Alphabet
IS08859-9	Latin Alphabet No. 5
IS08859-10	Latin Alphabet No. 6
IS08859-11	Thai, TIS 620-2569 character set
IS08859-13	Latin Alphabet No. 7
IS08859-14	Latin Alphabet No. 8
IS08859-15	Latin Alphabet No. 9
IS08859-16	Latin Alphabet No. 10
CP1250	Microsoft Windows Codepage 1250 (EE)
CP1251	Microsoft Windows Codepage 1251 (Cyril)
CP1252	Microsoft Windows Codepage 1252 (ANSI)
CP1253	Microsoft Windows Codepage 1253 (Greek)
CP1254	Microsoft Windows Codepage 1254 (Turk)
CP1255	Microsoft Windows Codepage 1255 (Hebr)
CP1256	Microsoft Windows Codepage 1256 (Arab)
CP1257	Microsoft Windows Codepage 1257 (BaltRim)
CP1258	Microsoft Windows Codepage 1258 (Viet)

KOI8-R Russian Net Character Set

The following multi-byte encodings are available in the plugin:

GB-EUC-H EUC-CN encoding

GB-EUC-V Vertical writing version of GB-EUC-H

GBK-EUC-H

Microsoft Code Page 936 (lfCharSet 0x86) GBK encoding

GBK-EUC-V

Vertical writing version of GBK-EUC-H

ETen-B5-H

Microsoft Code Page 950 (lfCharSet 0x88) Big Five character set with ETen extensions

ETen-B5-V

Vertical writing version of ETen-B5-H

90ms-RKSJ-H

Microsoft Code Page 932, JIS X 0208 character

90ms-RKSJ-V

Vertical writing version of 90ms-RKSJ-H

90msp-RKSJ-H

Microsoft Code Page 932, JIS X 0208 character (proportional)

EUC-H JIS X 0208 character set, EUC-JP encoding

EUC-V Vertical writing version of EUC-H

KSC-EUC-H

KS X 1001:1992 character set, EUC-KR encoding

KSC-EUC-V

Vertical writing version of KSC-EUC-H

KSCms-UHC-H

Microsoft Code Page 949 (lfCharSet 0x81), KS X 1001:1992 character set plus 8822 additional hangul, Unified Hangul Code (UHC) encoding (proportional)

KSCms-UHC-HW-H

Microsoft Code Page 949 (lfCharSet 0x81), KS X 1001:1992 character set plus 8822 additional hangul, Unified Hangul Code (UHC) encoding (fixed width)

KSCms-UHC-HW-V

Vertical writing version of KSCms-UHC-HW-H

UTF-8 UTF-8 encoding.

A Hollywood script has to invoke one of the following functions before using multi-byte encodings:

`doc:UseCNSEncodings()`

It makes simplified Chinese encodings (GB-EUC-H, GB-EUC-V, GBK-EUC-H, GBK-EUC-V) become available.

doc:UseCNTEncodings()

Makes traditional Chinese encodings (ETen-B5-H, ETen-B5-V) become available.

doc:UseJPEncodings()

Makes Japanese encodings (90ms-RKSJ-H, 90ms-RKSJ-V, 90msp-RKSJ-H, EUC-H, EUC-V) become available.

doc:UseKREncodings()

Makes Korean encodings (KSC-EUC-H, KSC-EUC-V, KSCms-UHC-H, KSCms-UHC-HW-H, KSCms-UHC-HW-V) become available.

doc:UseUTFEncodings()

Makes UTF-8 encoding become available.

5 Tutorial

5.1 Tutorial

This tutorial will teach you how to create your first PDF document with the plugin. The PDF document will contain two pages, one with a circle and one with a "Hello World" text.

First, you need to create a document object. This is done by calling `pdf.CreateDocument()` which creates a document object for you. The document object handle which is returned by `pdf.CreateDocument()` is then used in the following steps.

```
doc = pdf.CreateDocument()
```

As a second step you can set some document attributes. For example, here we set compression, encryption, page mode, and a password:

```
; set compression mode
doc:SetCompressionMode(#HPDF_COMP_ALL)

; set page mode to use outlines
doc:SetPageMode(#HPDF_PAGE_MODE_USE_OUTLINE)

; set password
doc:SetPassword("owner", "user")
```

After setting document attributes call `doc.AddPage()` to add a page to the document. The page handle returned is used in later operations on the page.

```
page1 = doc.AddPage()
```

To insert a new page before an existing page, `doc.InsertPage()`. For example, to insert `page0` before `page1`, do the following:

```
page0 = doc.InsertPage(page1)
```

After creating a new page, you can set some page attributes if necessary. Here we set the page size to B5 and the orientation to landscape:

```
page1:SetSize(#HPDF_PAGE_SIZE_B5, #HPDF_PAGE_LANDSCAPE)
```

Now that we have set up everything we can start adding content to the page. For example, this is how we add a "Hello World" text to the page:

```
font = doc:GetFont("Times-Roman")
page0:SetFontAndSize(font, 24)
page0:BeginText()
page0:TextOut(60, 60, "Hello World!")
page0:EndText()
```

We can also draw graphics primitives to the page, for example a filled circle:

```
page1:SetRGBFill(1.0, 0, 0)
page1:MoveTo(100, 100)
page1:LineTo(100, 180)
page1:Circle(100, 100, 80)
page1:Fill()
```

When you're done adding content to your pages, you'll probably want to save the PDF document to disk. This is possible by using the `doc:SaveToFile()` function. Here is how to save our PDF document:

```
doc:SaveToFile("test.pdf")
```

Now that we are finished, we have to free all resources belonging to the document object. This is done by calling the `doc:Free()` method, like so:

```
doc:Free()
```

Note that now that we have freed the document and all of its resources, we must no longer use any handles belonging to this document. In our case this means that we must no longer access the following handles: `doc`, `page0`, `page1`, and `font`. Thus, it is a good idea to set them to `Nil` so that Hollywood's garbage collector can kill them:

```
doc = Nil
page0 = Nil
page1 = Nil
font = Nil
```

Of course, you can also declare them as local variables and then they will be eaten by the garbage collector automatically once they become inaccessible.

That's it, congratulations, you have just created your first PDF document with the PDF plugin!

6 General functions

6.1 pdf.CloseDocument

NAME

pdf.CloseDocument – close PDF document

SYNOPSIS

```
pdf.CloseDocument(id)
```

FUNCTION

This function closes a document opened using `pdf.OpenDocument()` and frees all of its resources.

Note that this function must only be used for documents opened using `pdf.OpenDocument()`. Documents created using `pdf.CreateDocument()` must be freed using the `doc:Free()` method.

Also note that `pdf.CloseDocument()` must not be called before all vector brushes obtained via `pdf.GetBrush()` from the document have been freed.

INPUTS

id identifier of the PDF document to be closed

6.2 pdf.CreateDocument

NAME

pdf.CreateDocument – create a new PDF document

SYNOPSIS

```
doc = pdf.CreateDocument()
```

FUNCTION

`pdf.CreateDocument()` creates a new document object and returns its handle. You can then use all documents methods with this handle. On failure, `Nil` is returned.

When you're done with your document, don't forget to call `doc:Free()` on it to free all of its resources.

INPUTS

none

RESULTS

doc handle to a document

6.3 pdf.DeviceToPage

NAME

pdf.DeviceToPage – convert screen coordinates to page coordinates (V1.2)

SYNOPSIS

```
x, y = pdf.DeviceToPage(id, page, startx, starty, sizex, sizey, rotate,
                        devicex, devicey)
```

FUNCTION

This function can be used to convert the screen coordinates of the point specified by `devicex` and `devicey` to page coordinates.

The `rotate` argument can be used to specify the page orientation. This can be set to the following special values:

- 0: Normal.
- 1: Rotated 90 degrees clockwise.
- 2: Rotated 180 degrees.
- 3: Rotated 90 degrees counter-clockwise.

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

The page coordinate system has its origin at the left-bottom corner of the page, with the X-axis on the bottom going to the right, and the Y-axis on the left side going up. Note that this coordinate system can be altered when you zoom, scroll, or rotate a page, however, a point on the page should always have the same coordinate values in the page coordinate system.

The device coordinate system is device dependent. For screen devices, its origin is at the left-top corner of the window.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to use (starting from 1)
<code>startx</code>	left pixel position of the display area in device coordinates
<code>starty</code>	top pixel position of the display area in device coordinates
<code>sizex</code>	horizontal size (in pixels) for displaying the page
<code>sizey</code>	vertical size (in pixels) for displaying the page
<code>rotate</code>	page orientation (see above for possible values)
<code>devicex</code>	x value in device coordinates to be converted
<code>devicey</code>	y value in device coordinates to be converted

RESULTS

<code>x</code>	converted x value in page coordinates
<code>y</code>	converted y value in page coordinates

6.4 pdf.FindNext

NAME

pdf.FindNext – find next instance of search string (V1.1)

SYNOPSIS

```
res = pdf.FindNext(id, page)
```

FUNCTION

This function can be used to continue a search operation initiated by `pdf.FindStart()`. Specifically, `pdf.FindNext()` will find the next occurrence of the search string passed to `pdf.FindStart()`. If another instance of the search string could be found, `pdf.FindNext()` will return `True` and you can get the information about where the string was found using `pdf.GetFindResult()`. Otherwise, `False` is returned.

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to search (starting from 1)

RESULTS

<code>res</code>	<code>True</code> if the search string could be found, <code>False</code> otherwise
------------------	-------------------------------------------------------------------------------------

6.5 pdf.FindPrev

NAME

pdf.FindPrev – find previous instance of search string (V1.1)

SYNOPSIS

```
res = pdf.FindPrev(id, page)
```

FUNCTION

This function can be used to continue a search operation initiated by `pdf.FindStart()`. Specifically, `pdf.FindPrev()` will find the previous occurrence of the search string passed to `pdf.FindStart()`. If another instance of the search string could be found, `pdf.FindPrev()` will return `True` and you can get the information about where the string was found using `pdf.GetFindResult()`. Otherwise, `False` is returned.

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to search (starting from 1)

RESULTS

res True if the search string could be found, **False** otherwise

6.6 pdf.FindStart**NAME**

`pdf.FindStart` – initiate search operation (V1.1)

SYNOPSIS

`pdf.FindStart(id, page, s$[, flags, idx])`

FUNCTION

This function can be used to start a new search operation on the page specified by **page** in the document specified by **id**. You have to pass the string that the page should be searched for in the **s\$** argument. The optional argument **flags** can be used to configure additional options for the search operation. The **flags** parameter can be a combination of the following special constants:

#PDFFIND_MATCHCASE:

If this flag is set, the search operation will be done in a case-sensitive way.

#PDFFIND_MATCHWHOLEWORD:

If this flag is set, a search result is only triggered if **s\$** matches a whole word.

By default, the search operation starts at the beginning of the page. You can change this by passing a character index to start the search at in the optional **idx** parameter. Note that character indices start at 0. Passing -1 in the **idx** parameter will start the search at the end of the page.

The page to use must be specified in the **page** argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the **text** argument set to **True**. The PDF document specified by **id** must have been previously opened using `pdf.OpenDocument()`.

After you have called `pdf.FindStart()` to initiate the search operation, you then have to call either `pdf.FindNext()` or `pdf.FindPrev()` to actually execute the search operation.

INPUTS

id identifier of the PDF document to use

page page number to search (starting from 1)

s\$ string to search for

flags optional: combination of flags specifying additional options (see above) (defaults to 0)

idx optional: character index to start search at (defaults to 0)

6.7 pdf.FreePage

NAME

pdf.FreePage – free PDF document page (V1.1)

SYNOPSIS

```
pdf.FreePage(id, page)
```

FUNCTION

This function can be used to free a PDF document page loaded by `pdf.LoadPage()`. You have to pass the identifier of the PDF document to use in the `id` argument and the page number to free in the `page` argument. The page number must be in the range of 1 to the total number of pages in the document. The PDF document specified by `id` must have been opened using `pdf.OpenDocument()` before.

INPUTS

`id` identifier of the PDF document to use
`page` page number to free (starting from 1)

6.8 pdf.GetBookmarks

NAME

pdf.GetBookmarks – get all bookmarks in a document (V1.1)

SYNOPSIS

```
t = pdf.GetBookmarks(id)
```

FUNCTION

This function can be used to get all bookmarks in the PDF document specified by `id`. This PDF document must have been opened using `pdf.OpenDocument()`.

On return, `pdf.GetBookmarks()` will generate a table containing all bookmarks in the document. For each entry, the table will have the following fields initialized:

Title: The bookmark's title text.

Action: This field specifies what should happen if the respective bookmark is clicked. This will be set to one of the following special constants:

#PDFACTION_GOTO:
 Skip to page in current document.

#PDFACTION_REMOTEGOTO:
 Skip to page in another document.

#PDFACTION_URI:
 Open an URI.

#PDFACTION_LAUNCH:
 Launch a program.

#PDFACTION_UNSUPPORTED:
 Unknown action.

Target: This will be set to the bookmark's target. Depending on **Action**, this may be set to a page number, a URI, or the path to an external file.

Children: If the bookmark can be unfolded, this item will be set to another table containing the same elements as its parent. Bookmarks can be infinitely nested.

INPUTS

id identifier of the PDF document to use

RESULTS

t table containing all document bookmarks (see above)

6.9 pdf.GetBoundedText

NAME

pdf.GetBoundedText – get text within bounding rectangle (V1.1)

SYNOPSIS

```
t$ = pdf.GetBoundedText(id, page, left, top, right, bottom)
```

FUNCTION

This function can be used to extract the text that is within the bounding rectangle specified by **left**, **top**, **right**, and **bottom** from a page. If there is no text within the specified bounding rectangle, an empty string is returned.

The page to use must be specified in the **page** argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the **text** argument set to **True**. The PDF document specified by **id** must have been previously opened using `pdf.OpenDocument()`.

INPUTS

id identifier of the PDF document to use

page page number to use (starting from 1)

left left boundary

top top boundary

right right boundary

bottom bottom boundary

RESULTS

t\$ the text within the bounding rectangle

6.10 pdf.GetBrush

NAME

pdf.GetBrush – get PDF page as vector brush

SYNOPSIS

```
[id, t] = pdf.GetBrush(id, page, brid[, transparent, getlinks])
```

FUNCTION

This function can be used to convert a page from the PDF document specified by `id` to a vector brush using the identifier `brid`. If you pass `Nil` in `brid`, `pdf.GetBrush()` will automatically choose a vacant identifier and return it.

The page to convert must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document. The PDF document specified by `id` must have been opened using `pdf.OpenDocument()`.

The optional parameter `transparent` allows you to specify whether you'd like the page background to be transparent or white. If you pass `True` here, you'll get a vector brush in which the page background is completely transparent by using alpha channel transparency. Otherwise the page background will be white and your vector brush won't use any transparency.

Note that the vector brush will still depend on the PDF document so it is not allowed to call `pdf.CloseDocument()` on the document while you still need the brush.

Also note that you should only use this function for pages that haven't been loaded with `pdf.LoadPage()` before. If you want to convert a page that has been loaded using `pdf.LoadPage()` to a brush, use the `pdf.GetBrushFromPage()` function instead. See Section 6.11 [`pdf.GetBrushFromPage`], page 28, for details.

Starting with version 1.1, there is an optional argument called `getlinks`. If this is set to `True`, `pdf.GetBrush()` will return a table containing all links in the page. The table is returned as the second return value if `getlinks` is set to `True`. For each entry, the table will have the following fields initialized:

Action: This field specifies what should happen if the respective link is clicked. This will be set to one of the following special constants:

```
#PDFACTION_GOTO:
    Skip to page in current document.

#PDFACTION_REMOTEGOTO:
    Skip to page in another document.

#PDFACTION_URI:
    Open an URI.

#PDFACTION_LAUNCH:
    Launch a program.

#PDFACTION_UNSUPPORTED:
    Unknown action.
```

Target: This will be set to the link's target. Depending on **Action**, this may be set to a page number, a URI, or the path to an external file.

Left: Left edge of the link's bounding rectangle.
Top: Top edge of the link's bounding rectangle.
Right: Right edge of the link's bounding rectangle.
Bottom: Bottom edge of the link's bounding rectangle.

INPUTS

id identifier of the PDF document to use
page page number to convert (starting from 1)
brid identifier for the vector brush or Nil for auto id selection
transparent optional: True for a transparent page background, False for a white page background
getlinks optional: True if page links should be returned (see above) (defaults to False) (V1.1)

RESULTS

id optional: identifier of the brush; will only be returned when you pass Nil as argument 3 (see above)
t optional: table containing all page links (see above) (V1.1)

6.11 pdf.GetBrushFromPage

NAME

pdf.GetBrushFromPage – get PDF page as vector brush (V1.1)

SYNOPSIS

```
[id] = pdf.GetBrushFromPage(id, page, brid[, transparent])
```

FUNCTION

This function can be used to convert a page from the PDF document specified by **id** to a vector brush using the identifier **brid**. If you pass Nil in **brid**, **pdf.GetBrush()** will automatically choose a vacant identifier and return it.

The page to convert must be specified in the **page** argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using **pdf.LoadPage()**. The PDF document specified by **id** must have been previously opened using **pdf.OpenDocument()**.

The optional parameter **transparent** allows you to specify whether you'd like the page background to be transparent or white. If you pass True here, you'll get a vector brush in which the page background is completely transparent by using alpha channel transparency. Otherwise the page background will be white and your vector brush won't use any transparency.

Note that the vector brush will still depend on the PDF document so it is not allowed to call **pdf.CloseDocument()** on the document before freeing the brush. It also is not allowed to call **pdf.FreePage()** before freeing the brush.

If you want to convert a PDF page into a brush without being forced to load the page using `pdf.LoadPage()` first, use the `pdf.GetBrush()` function. See [Section 6.10 \[pdf.GetBrush\]](#), page 27, for details.

INPUTS

`id` identifier of the PDF document to use

`page` page number to convert (starting from 1)

`brid` identifier for the vector brush or `Nil` for auto id selection

`transparent` optional: `True` for a transparent page background, `False` for a white page background

RESULTS

`id` optional: identifier of the brush; will only be returned when you pass `Nil` as argument 3 (see above)

6.12 pdf.GetCharBox

NAME

`pdf.GetCharBox` – get bounding rectangle of character (V1.1)

SYNOPSIS

```
left, top, right, bottom = pdf.GetCharBox(id, page, idx)
```

FUNCTION

This function can be used to get the bounding box of the character at index `idx` on the page specified by `page`. Note that character indices start at 0 whereas page indices start at 1.

The page specified in the `page` argument must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

`id` identifier of the PDF document to use

`page` page number to use (starting from 1)

`idx` index of character whose bounding rectangle to retrieve (starting from 0)

RESULTS

`left` left boundary

`top` top boundary

`right` right boundary

`bottom` bottom boundary

6.13 pdf.GetCharIndexAtPos

NAME

pdf.GetCharIndexAtPos – get character at page position (V1.1)

SYNOPSIS

```
idx = pdf.GetCharIndexAtPos(id, page, x, y[, xt, yt])
```

FUNCTION

This function can be used to get the index of a character at or nearby the position specified by `x` and `y` on the page. The optional `xt` and `yt` parameters can be used to specify a tolerance value (in point units) that should be used when getting the character. The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`. `pdf.GetCharIndexAtPos()` will return the zero-based index of the character at, or nearby the point specified by `x` and `y`. If there is no character at or nearby the point, the return value will be -1. If an error occurs, -3 will be returned.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to use (starting from 1)
<code>x</code>	x position to use
<code>y</code>	y position to use
<code>xt</code>	optional: x tolerance value (defaults to 0)
<code>yt</code>	optional: y tolerance value (defaults to 0)

RESULTS

<code>idx</code>	index of character at the specified point or -1 or -3 (see above)
------------------	-------------------------------------------------------------------

6.14 pdf.GetCharOrigin

NAME

pdf.GetCharOrigin – get origin of character (V1.1)

SYNOPSIS

```
x, y = pdf.GetCharOrigin(id, page, idx)
```

FUNCTION

This function can be used to get the origin of the character at the index specified by `idx` (starting at 0).

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to use (starting from 1)
<code>idx</code>	character index to use (starting from 0)

RESULTS

<code>x</code>	x position of origin
<code>y</code>	y position of origin

6.15 pdf.GetCropBox

NAME

`pdf.GetCropBox` – get crop box from page dictionary (V1.2)

SYNOPSIS

```
left, top, right, bottom = pdf.GetCropBox(id, page)
```

FUNCTION

This function can be used to get the "CropBox" entry from the page dictionary. The page specified in the `page` argument must have been previously loaded using `pdf.LoadPage()`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to use (starting from 1)

RESULTS

<code>left</code>	left boundary
<code>top</code>	top boundary
<code>right</code>	right boundary
<code>bottom</code>	bottom boundary

6.16 pdf.GetFindResult

NAME

`pdf.GetFindResult` – get result of search operation (V1.1)

SYNOPSIS

```
idx, len = pdf.GetFindResult(id, page)
```

FUNCTION

This function can be used to get the result of a search operation after `pdf.FindNext()` or `pdf.FindPrev()` has returned `True`. In that case, `pdf.GetFindResult()` will return

the character index of the search string's occurrence on the page as well as its length. Character indices start from 0.

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

`id` identifier of the PDF document to use

`page` page number to use (starting from 1)

RESULTS

`idx` start offset of next occurrence of search string on page

`len` length of search string

6.17 pdf.GetLastError

NAME

`pdf.GetLastError` – get last error code (V1.1)

SYNOPSIS

```
error = pdf.GetLastError()
```

FUNCTION

If `pdf.OpenDocument()` fails, `pdf.GetLastError()` can be used to get additional information why the document couldn't be opened. This is especially useful to find out if the document couldn't be opened because it is password-protected.

`pdf.GetLastError()` will return one of the following error codes:

`#PDFERR_SUCCESS:`

No error occurred.

`#PDFERR_UNKNOWN:`

An unknown error occurred.

`#PDFERR_FILE:`

The file couldn't be found.

`#PDFERR_FORMAT:`

The file format couldn't be recognized.

`#PDFERR_PASSWORD:`

The PDF document is password-protected.

`#PDFERR_SECURITY:`

Security settings forbid opening of this document.

`#PDFERR_PAGE:`

The page table is corrupted.

Note that you have to call `pdf.GetLastError()` immediately after `pdf.OpenDocument()` to get the correct result code.

INPUTS

none

RESULTS

error last error code

6.18 pdf.GetMediaBox

NAME

`pdf.GetMediaBox` – get media box from page dictionary (V1.2)

SYNOPSIS

`left, top, right, bottom = pdf.GetMediaBox(id, page)`

FUNCTION

This function can be used to get the "MediaBox" entry from the page dictionary. The page specified in the `page` argument must have been previously loaded using `pdf.LoadPage()`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

`id` identifier of the PDF document to use
`page` page number to use (starting from 1)

RESULTS

`left` left boundary
`top` top boundary
`right` right boundary
`bottom` bottom boundary

6.19 pdf.GetMetaText

NAME

`pdf.GetMetaText` – get meta text from document (V1.1)

SYNOPSIS

`t$ = pdf.GetMetaText(id, attr$)`

FUNCTION

This function can be used to get meta text from the PDF document specified by `id`. This PDF document must have been opened using `pdf.OpenDocument()`. The `attr$` argument specifies which text to get. This must be a string and can be set to the following values:

Title: Document's title.

Author: Document's author.
Subject: Document's subject.
Keywords:
 Keywords.
Creator Document's creator.
Producer:
 Document's producer.
CreationDate
 Document's creation date.
ModDate: Document's last modification date.

Note that meta texts aren't always set. If there is no meta text for the specified attribute, an empty string is returned.

INPUTS

id identifier of the PDF document to use
attr\$ string specifying the meta data to get (see above for possible values)

RESULTS

t\$ meta data retrieved from document

6.20 pdf.GetObjectType

NAME

pdf.GetObjectType – get PDF document object type

SYNOPSIS

```
type = pdf.GetObjectType()
```

FUNCTION

This function returns the object type used by PDF documents loaded using the `pdf.OpenDocument()` function. You can then use this object type with functions from Hollywood's object library such as `GetAttribute()`, `SetObjectData()`, `GetObjectData()`, etc.

In particular, Hollywood's `GetAttribute()` function may be used to query certain properties of PDF documents loaded using `pdf.OpenDocument()`. The following attributes are currently supported by `GetAttribute()` for PDF documents:

#PDFATTRPAGES:

Returns the number of pages in the document.

#PDFATTRVERSION:

Returns the PDF version this document uses. This will be an integer number, e.g. 14 for 1.4, 15 for 1.5, etc. (V1.1)

#PDFATTRPERMISSIONS:

Returns a 32-bit integer describing the document's permission flags. Please refer to the PDF Reference for detailed descriptions on permissions. (V1.1)

INPUTS

none

RESULTS

type internal PDF document type for use with Hollywood's object library

EXAMPLE

```
pdf.OpenDocument(1, "test.pdf")
PDF_DOCUMENT = pdf.GetObjectType()
numpages = GetAttribute(PDF_DOCUMENT, 1, #PDFATTRPAGES)
```

The code above opens `test.pdf` and queries the number of pages in the document via `GetAttribute()`.

6.21 pdf.GetPageLabel

NAME

`pdf.GetPageLabel` – get page label text (V1.1)

SYNOPSIS

```
l$ = pdf.GetPageLabel(id, page)
```

FUNCTION

This function can be used to get the label of the page specified by the `page` argument. This must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

Note that page labels aren't always set. If there is no label for the page, an empty string is returned.

INPUTS

id identifier of the PDF document to use

page page number to use (starting from 1)

RESULTS

l\$ page's label

6.22 pdf.GetPageLen

NAME

`pdf.GetPageLen` – get number of characters on page (V1.1)

SYNOPSIS

```
len = pdf.GetPageLen(id, page)
```

FUNCTION

This function can be used to get the number of characters on the page specified by the `page` argument. This must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()`

with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

`id` identifier of the PDF document to use
`page` page number to use (starting from 1)

RESULTS

`len` number of characters on page

6.23 pdf.GetPageLinks

NAME

`pdf.GetPageLinks` – get all links on a PDF page (V1.1)

SYNOPSIS

```
t = pdf.GetPageLinks(id, page)
```

FUNCTION

This function can be used to get all links from a PDF page. The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and it must have been loaded using `pdf.LoadPage()`. The PDF document specified by `id` must have been opened using `pdf.OpenDocument()`.

On return, `pdf.GetPageLinks()` will generate a table containing all links in the page. For each entry, the table will have the following fields initialized:

Action: This field specifies what should happen if the respective link is clicked. This will be set to one of the following special constants:

#PDFACTION_GOTO:
Skip to page in current document.

#PDFACTION_REMOTEGOTO:
Skip to page in another document.

#PDFACTION_URI:
Open an URI.

#PDFACTION_LAUNCH:
Launch a program.

#PDFACTION_UNSUPPORTED:
Unknown action.

Target: This will be set to the link's target. Depending on **Action**, this may be set to a page number, a URI, or the path to an external file.

Left: Left edge of the link's bounding rectangle.

Top: Top edge of the link's bounding rectangle.

Right: Right edge of the link's bounding rectangle.

Bottom: Bottom edge of the link's bounding rectangle.

INPUTS

id identifier of the PDF document to use
page page number to use (starting from 1)

RESULTS

t table containing all page links (see above)

6.24 pdf.GetRects

NAME

pdf.GetRects – get bounding rectangles of character range (V1.1)

SYNOPSIS

```
t = pdf.GetRects(id, page, idx, len)
```

FUNCTION

This function can be used to get a series of rectangles that encloses the text starting at the index specified by **idx**, spanning over **len** number of characters. Note that character indices start at 0. If you pass -1 in **len**, `pdf.GetRects()` will automatically extend the selection to all remaining characters.

The page to use must be specified in the **page** argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the **text** argument set to **True**. The PDF document specified by **id** must have been previously opened using `pdf.OpenDocument()`.

This function will return a table containing one subtable per bounding rectangle. Each of those subtables will have the following fields initialized:

Left: Left boundary.
Top: Top boundary.
Right: Right boundary.
Bottom: Bottom boundary.

INPUTS

id identifier of the PDF document to use
page page number to use (starting from 1)
idx character index to use (starting from 0)
len number of characters to use or -1 for all remaining characters

RESULTS

t table containing a series of bounding rectangles (see above)

6.25 pdf.GetText

NAME

pdf.GetText – get text on page (V1.1)

SYNOPSIS

```
t$ = pdf.GetText(id, page, idx, len)
```

FUNCTION

This function can be used to extract the text starting at the index specified by `idx` and spanning over `len` number of characters from a page. Note that character indices start at 0. If you pass -1 in `len`, `pdf.GetText()` will automatically extract all remaining characters after the specified index.

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()` with the `text` argument set to `True`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to use (starting from 1)
<code>idx</code>	character index to use (starting from 0)
<code>len</code>	number of characters to use or -1 for all remaining characters

RESULTS

<code>t\$</code>	text that has been extracted
------------------	------------------------------

6.26 pdf.GetVersion

NAME

pdf.GetVersion – get libHaru version

SYNOPSIS

```
ver$ = pdf.GetVersion()
```

FUNCTION

This function can be used to query the version of libHaru used by the plugin. It will return a version string.

INPUTS

none

RESULTS

<code>ver\$</code>	libHaru version string
--------------------	------------------------

6.27 pdf.IsPDF

NAME

pdf.IsPDF – check if file is a valid PDF document (V1.1)

SYNOPSIS

```
ok = pdf.IsPDF(f$)
```

FUNCTION

This function checks if the file specified by `f$` is in the PDF format and returns `True` if it is, `False` otherwise.

INPUTS

`f$` file to check

RESULTS

`ok` `True` if the specified file is a PDF document

6.28 pdf.LoadPage

NAME

pdf.LoadPage – load page from PDF document (V1.1)

SYNOPSIS

```
pdf.LoadPage(id, page[, loadtext])
```

FUNCTION

This function can be used to load a page from the PDF document specified by `id`. The page to load must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document. The PDF document specified by `id` must have been opened using `pdf.OpenDocument()` before.

If the optional argument `loadtext` is set to `True`, `pdf.LoadPage()` will also load the page's text. This is necessary if you want to use functions that deal with text on a PDF page, e.g. `pdf.GetText()` or `pdf.FindStart()`.

When you're done with the page, you should call `pdf.FreePage()` to free its resources. This is also done automatically when calling `pdf.CloseDocument()`. See [Section 6.7 \[pdf:FreePage\]](#), [page 25](#), for details.

INPUTS

`id` identifier of the PDF document to use

`page` page number to load (starting from 1)

`loadtext` optional: `True` if the page's text should be loaded (defaults to `False`)

6.29 pdf.OpenDocument

NAME

pdf.OpenDocument – open PDF document

SYNOPSIS

```
[id] = pdf.OpenDocument(id, file$[, t])
```

FUNCTION

This function opens an existing PDF document which is specified by `file$` and assigns the identifier `id` to it. If you pass `Nil` in `id`, `pdf.OpenDocument()` will automatically choose a vacant identifier and return it.

The optional table argument allows you to configure further options:

Password:

If the document is password-protected, you can specify the password needed to open this document here.

Adapter: This tag allows you to specify one or more file adapters that should be asked to open the specified file. This must be set to a string containing the name(s) of one or more adapter(s). Defaults to `default`. See your Hollywood manual for more information on file adapters.

If `pdf.OpenDocument()` fails, `pdf.GetLastError()` can be used to get additional information why the document couldn't be opened. This is especially useful to find out if the document couldn't be opened because it is password-protected. See [Section 6.17 \[pdf.GetLastError\]](#), page 32, for details.

INPUTS

<code>id</code>	identifier for the PDF document or <code>Nil</code> for auto id selection
<code>file\$</code>	file to load
<code>table</code>	optional: table specifying further options (see above)

RESULTS

<code>id</code>	optional: identifier of the document; will only be returned when you pass <code>Nil</code> as argument 1 (see above)
-----------------	----------------------------------------------------------------------------------------------------------------------

6.30 pdf.PageToDevice

NAME

pdf.PageToDevice – convert page coordinates to screen coordinates (V1.2)

SYNOPSIS

```
x, y = pdf.PageToDevice(id, page, startx, starty, sizex, sizey, rotate,
                        pagex, pagey)
```

FUNCTION

This function can be used to convert the page coordinates of the point specified by `pagex` and `pagey` to screen coordinates.

The `rotate` argument can be used to specify the page orientation. This can be set to the following special values:

- 0: Normal.
- 1: Rotated 90 degrees clockwise.
- 2: Rotated 180 degrees.
- 3: Rotated 90 degrees counter-clockwise.

The page to use must be specified in the `page` argument. It must be a number in the range of 1 to the total number of pages in the document and the page must have been previously loaded using `pdf.LoadPage()`. The PDF document specified by `id` must have been previously opened using `pdf.OpenDocument()`.

The page coordinate system has its origin at the left-bottom corner of the page, with the X-axis on the bottom going to the right, and the Y-axis on the left side going up. Note that this coordinate system can be altered when you zoom, scroll, or rotate a page, however, a point on the page should always have the same coordinate values in the page coordinate system.

The device coordinate system is device dependent. For screen devices, its origin is at the left-top corner of the window.

INPUTS

<code>id</code>	identifier of the PDF document to use
<code>page</code>	page number to use (starting from 1)
<code>startx</code>	left pixel position of the display area in device coordinates
<code>starty</code>	top pixel position of the display area in device coordinates
<code>size_x</code>	horizontal size (in pixels) for displaying the page
<code>size_y</code>	vertical size (in pixels) for displaying the page
<code>rotate</code>	page orientation (see above for possible values)
<code>page_x</code>	x value in page coordinates
<code>page_y</code>	y value in page coordinates

RESULTS

<code>x</code>	x value in device coordinates
<code>y</code>	y value in device coordinates

7 Annotation methods

7.1 annot:SetBorderStyle

NAME

annot:SetBorderStyle – set appearance of text annotation

SYNOPSIS

```
status = annot:SetBorderStyle(subtype, width, dashon, dashoff, dashphase)
```

FUNCTION

`annot:SetBorderStyle()` defines the appearance of a text annotation. `subtype` must be one of the following constants:

`#HPDF_BS_SOLID:`

Solid rectangle

`#HPDF_BS_DASHED:`

Dashed rectangle

`#HPDF_BS_BEVELED:`

Embossed rectangle

`#HPDF_BS_INSET:`

Engraved rectangle

`#HPDF_BS_UNDERLINED:`

Single line under the bottom of the annotation

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`subtype` one of the constants listed above

`width` the width of an annotation's border

`dashon` the dash style

`dashoff` the dash style

`dashphase`
the dash style

RESULTS

`status` status code

7.2 annot:SetCMYKColor

NAME

annot:SetCMYKColor – set CMYK color

SYNOPSIS

```
status = annot:SetCMYKColor(cmyk)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `cmyk` parameter must be a table with the following fields initialized:

C	Cyan level of color.
Y	Yellow level of color.
M	Magenta level of color.
K	Black level of color.

All values must be between 0 and 1.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

<code>cmyk</code>	CMYK color
-------------------	------------

RESULTS

<code>status</code>	status code
---------------------	-------------

7.3 annot:SetFreeTextAnnot2PointCalloutLine

NAME

annot:SetFreeTextAnnot2PointCalloutLine – set free text annotation two point callout line

SYNOPSIS

```
status = annot:SetFreeTextAnnot2PointCalloutLine(startpoint, endpoint)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The parameters `startpoint` and `endpoint` must be tables that describe a point each. Thus, each of those tables must contain the fields `x` and `y`.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

<code>startpoint</code>	start point
<code>endpoint</code>	end point

RESULTS

`status` status code

7.4 annot:SetFreeTextAnnot3PointCalloutLine**NAME**

`annot:SetFreeTextAnnot3PointCalloutLine` – set free text annotation three point callout line

SYNOPSIS

```
status = annot:SetFreeTextAnnot3PointCalloutLine(startpoint, kneepoint,
                                                  endpoint)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The parameters `startpoint`, `kneepoint`, and `endpoint` must be tables that describe a point each. Thus, each of those tables must contain the fields `x` and `y`.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`startpoint`
 start point

`kneepoint`
 knee point

`endpoint` end point

RESULTS

`status` status code

7.5 annot:SetFreeTextAnnotDefaultStyle**NAME**

`annot:SetFreeTextAnnotDefaultStyle` – set free text annotation default style

SYNOPSIS

```
status = annot:SetFreeTextAnnotDefaultStyle(style)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`style` default style

RESULTS

`status` status code

7.6 annot:SetFreeTextAnnotLineEndingStyle

NAME

annot:SetFreeTextAnnotLineEndingStyle – set free text annotation line ending style

SYNOPSIS

```
status = annot:SetFreeTextAnnotLineEndingStyle(startstyle, endstyle)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The parameters `startstyle` and `endstyle` must be one of the following constants:

```
#HPDF_LINE_ANNOT_NONE
#HPDF_LINE_ANNOT_SQUARE
#HPDF_LINE_ANNOT_CIRCLE
#HPDF_LINE_ANNOT_DIAMOND
#HPDF_LINE_ANNOT_OPENARROW
#HPDF_LINE_ANNOT_CLOSEDARROW
#HPDF_LINE_ANNOT_BUTT
#HPDF_LINE_ANNOT_ROPENARROW
#HPDF_LINE_ANNOT_RCLOSEDARROW
#HPDF_LINE_ANNOT_SLASH
```

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

```
startstyle      start style
endstyle        end style
```

RESULTS

```
status          status code
```

7.7 annot:SetGrayColor

NAME

annot:SetGrayColor – set gray color

SYNOPSIS

```
status = annot:SetGrayColor(gray)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `gray` parameter must be between 0 and 1.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

```
gray           gray color
```

RESULTS

`status` status code

7.8 `annot:SetLineAnnotCaption`

NAME

`annot:SetLineAnnotCaption` – set line annotation caption

SYNOPSIS

```
status = annot:SetLineAnnotCaption(show, pos, horz, vert)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `pos` argument must be one of the following constants:

```
#HPDF_LINE_ANNOT_CAP_INLINE  
#HPDF_LINE_ANNOT_CAP_TOP
```

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`show` boolean value that indicates whether to show the caption
`pos` caption position (see above for possible values)
`horz` horizontal offset
`vert` vertical offset

RESULTS

`status` status code

7.9 `annot:SetLineAnnotLeader`

NAME

`annot:SetLineAnnotLeader` – set line annotation leader

SYNOPSIS

```
status = annot:SetLineAnnotLeader(len, extlen, offsetlen)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`len` length
`extlen` extended length

`offsetlen`
offset length

RESULTS

`status` status code

7.10 `annot:SetLineAnnotPosition`

NAME

`annot:SetLineAnnotPosition` – set line annotation position

SYNOPSIS

```
status = annot:SetLineAnnotPosition(startpoint, startstyle, endpoint,
                                     endstyle)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `startstyle` and `endstyle` parameters must be one of the following constants:

```
#HPDF_LINE_ANNOT_NONE
#HPDF_LINE_ANNOT_SQUARE
#HPDF_LINE_ANNOT_CIRCLE
#HPDF_LINE_ANNOT_DIAMOND
#HPDF_LINE_ANNOT_OPENARROW
#HPDF_LINE_ANNOT_CLOSEDARROW
#HPDF_LINE_ANNOT_BUTT
#HPDF_LINE_ANNOT_ROPENARROW
#HPDF_LINE_ANNOT_RCLOSEDARROW
#HPDF_LINE_ANNOT_SLASH
```

The parameters `startpoint` and `endpoint` must be tables that describe a point each. Thus, each of those tables must contain the fields `x` and `y`.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`startpoint`
start point

`startstyle`
start style (see above for possible values)

`endpoint` end point

`endstyle` end style (see above for possible values)

RESULTS

`status` status code

7.11 annot:SetLinkAnnotBorderStyle

NAME

annot:SetLinkAnnotBorderStyle – set annotation border style

SYNOPSIS

```
status = annot:SetLinkAnnotBorderStyle(width, dashon, dashoff)
```

FUNCTION

`annot:SetLinkAnnotBorderStyle()` defines the style of the annotation's border.

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`width` the width of an annotation's border
`dashon` the dash style
`dashoff` the dash style

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_ANNOTATION - An invalid annotation handle was set.
 #HPDF_INVALID_PARAMETER - An invalid width value was set.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

7.12 annot:SetLinkAnnotHighlightMode

NAME

annot:SetLinkAnnotHighlightMode – set highlight appearance

SYNOPSIS

```
status = annot:SetLinkAnnotHighlightMode(mode)
```

FUNCTION

`annot:SetLinkAnnotHighlightMode()` defines the appearance when a mouse clicks on a link annotation. `mode` can be one of the following constants:

#HPDF_ANNOT_NO_HIGHLIGHT
 No highlighting.

#HPDF_ANNOT_INVERT_BOX
 Invert the contents of the area of annotation.

#HPDF_ANNOT_INVERT_BORDER
 Invert the annotation's border.

#HPDF_ANNOT_DOWN_APPEARANCE
 Dent the annotation.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`mode` one of the constants listed above

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_ANNOTATION` - An invalid annotation handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

7.13 `annot:SetMarkupAnnotCloudEffect`

NAME

`annot:SetMarkupAnnotCloudEffect` – set markup annotation cloud effect

SYNOPSIS

```
status = annot:SetMarkupAnnotCloudEffect(cloudintensity)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`cloudintensity`
 cloud effect

RESULTS

`status` status code

7.14 `annot:SetMarkupAnnotCreationDate`

NAME

`annot:SetMarkupAnnotCreationDate` – set markup annotation creation date

SYNOPSIS

```
status = annot:SetMarkupAnnotCreationDate(value)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

`value` must be a table containing a datetime description. The table must contain the following fields:

Day: Between 1 and 31 (depends on the month).

Month: Between 1 and 12.

Year: The year.

Hour: Between 0 and 23.

Minutes: Between 0 and 59.

Seconds: Between 0 and 59.

Ind: Relationship of local time to Universal Time. This can be " ", "+", "-", or "Z".

Off_Hour:
If ind is not space, 0 to 23 is valid. Otherwise, ignored.

Off_Minutes:
If ind is not space, 0 to 59 is valid. Otherwise, ignored.

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

value datetime description

RESULTS

status status code

7.15 annot:SetMarkupAnnotIntent**NAME**

annot:SetMarkupAnnotIntent – set markup annotation intent

SYNOPSIS

```
status = annot:SetMarkupAnnotIntent(intent)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `intent` parameter must be one of the following constants:

```
#HPDF_ANNOT_INTENT_FREETEXTCALLOUT
#HPDF_ANNOT_INTENT_FREETEXTTYPEWRITER
#HPDF_ANNOT_INTENT_LINEARROW
#HPDF_ANNOT_INTENT_LINEDIMENSION
#HPDF_ANNOT_INTENT_POLYGONCLOUD
#HPDF_ANNOT_INTENT_POLYLINEDIMENSION
#HPDF_ANNOT_INTENT_POLYGONDIMENSION
```

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

intent desired intent (see above for possible values)

RESULTS

status status code

7.16 annot:SetMarkupAnnotInteriorCMYKColor

NAME

annot:SetMarkupAnnotInteriorCMYKColor – set markup annotation interior CMYK color

SYNOPSIS

```
status = annot:SetMarkupAnnotInteriorCMYKColor(cmyk)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The cmyk parameter must be a table with the following fields initialized:

- C Cyan level of color.
- Y Yellow level of color.
- M Magenta level of color.
- K Black level of color.

All fields must contain values between 0 and 1.

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

cmyk CMYK color as a table

RESULTS

status status code

7.17 annot:SetMarkupAnnotInteriorGrayColor

NAME

annot:SetMarkupAnnotInteriorGrayColor – set markup annotation interior gray color

SYNOPSIS

```
status = annot:SetMarkupAnnotInteriorGrayColor(gray)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

gray gray color

RESULTS

status status code

7.18 annot:SetMarkupAnnotInteriorRGBColor

NAME

annot:SetMarkupAnnotInteriorRGBColor – set markup annotation interior RGB color

SYNOPSIS

```
status = annot:SetMarkupAnnotInteriorRGBColor(rgb)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `rgb` parameter must be a table containing the following fields:

R Red level of color.

G Green level of color.

B Blue level of color.

All fields must be values between 0 and 1.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`rgb` RGB color as a table

RESULTS

`status` status code

7.19 annot:SetMarkupAnnotInteriorTransparent

NAME

annot:SetMarkupAnnotInteriorTransparent – set markup annotation interior transparent

SYNOPSIS

```
status = annot:SetMarkupAnnotInteriorTransparent()
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

7.20 annot:SetMarkupAnnotPopup

NAME

annot:SetMarkupAnnotPopup – set markup annotation popup

SYNOPSIS

```
status = annot:SetMarkupAnnotPopup(popup)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

popup annotation object to be used as popup

RESULTS

status status code

7.21 annot:SetMarkupAnnotQuadPoints

NAME

annot:SetMarkupAnnotQuadPoints – set markup annotation quad points

SYNOPSIS

```
status = annot:SetMarkupAnnotQuadPoints(lb, rb, rt, lt)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The parameters `lb`, `rb`, `rt`, and `lt` must be tables that describe a point each. Thus, each of those tables must contain the fields `x` and `y`.

Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

lb left bottom point
rb right bottom point
rt right top point
lt left top point

RESULTS

status status code

7.22 annot:SetMarkupAnnotRectDiff

NAME

annot:SetMarkupAnnotRectDiff – set markup annotation rect diff

SYNOPSIS

```
status = annot:SetMarkupAnnotRectDiff(rect)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`rect` rect diff

RESULTS

`status` status code

7.23 annot:SetMarkupAnnotSubject

NAME

annot:SetMarkupAnnotSubject – set markup annotation subject

SYNOPSIS

```
status = annot:SetMarkupAnnotSubject(subj)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`subj` subject for markup annotation

RESULTS

`status` status code

7.24 annot:SetMarkupAnnotTitle

NAME

annot:SetMarkupAnnotTitle – set markup annotation title

SYNOPSIS

```
status = annot:SetMarkupAnnotTitle(name)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.
Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`name` title for markup annotation

RESULTS

`status` status code

7.25 `annot:SetMarkupAnnotTransparency`

NAME

`annot:SetMarkupAnnotTransparency` – set markup annotation transparency

SYNOPSIS

`status = annot:SetMarkupAnnotTransparency(value)`

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.
Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`value` transparency setting

RESULTS

`status` status code

7.26 `annot:SetNoColor`

NAME

`annot:SetNoColor` – set no color

SYNOPSIS

`status = annot:SetNoColor()`

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors.
Returns #HPDF_OK on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

7.27 annot:SetPopupAnnotOpened

NAME

annot:SetPopupAnnotOpened – set visibility state of popup annotation

SYNOPSIS

```
status = annot:SetPopupAnnotOpened(open)
```

FUNCTION

`annot:SetPopupAnnotOpened()` defines whether the popup annotation is initially open. Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`open` True means the annotation initially displayed open

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_ANNOTATION` - An invalid annotation handle was set.
`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

7.28 annot:SetRGBColor

NAME

annot:SetRGBColor – set RGB color

SYNOPSIS

```
status = annot:SetRGBColor(rgb)
```

FUNCTION

This method is currently undocumented in libHaru. Complain to the libHaru authors. The `rgb` parameter must be a table with the following fields initialized:

`R` Red level of color.
`G` Green level of color.
`B` Blue level of color.

All values must be between 0 and 1.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`rgb` RGB color

RESULTS

`status` status code

7.29 annot:SetTextAnnotIcon

NAME

annot:SetTextAnnotIcon – set annotation icon

SYNOPSIS

```
status = annot:SetTextAnnotIcon(icon)
```

FUNCTION

`annot:SetTextAnnotIcon()` defines the style of the annotation's icon. `icon` can be one of the following constants:

```
#HPDF_ANNOT_ICON_COMMENT
#HPDF_ANNOT_ICON_KEY
#HPDF_ANNOT_ICON_NOTE
#HPDF_ANNOT_ICON_HELP
#HPDF_ANNOT_ICON_NEW_PARAGRAPH
#HPDF_ANNOT_ICON_PARAGRAPH
#HPDF_ANNOT_ICON_INSERT
```

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`icon` one of the constants listed above

RESULTS

`status` status code

ERRORS

```
#HPDF_INVALID_ANNOTATION - An invalid annotation handle was set.
#HPDF_ANNOT_INVALID_ICON - An invalid icon-style was specified.
#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
```

7.30 annot:SetTextAnnotOpened

NAME

annot:SetTextAnnotOpened – set visibility state of text annotation

SYNOPSIS

```
status = annot:SetTextAnnotOpened(open)
```

FUNCTION

`annot:SetTextAnnotOpened()` defines whether the text-annotation is initially open.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

`open` True means the annotation initially displayed open

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_ANNOTATION - An invalid annotation handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

8 Destination methods

8.1 dest:SetFit

NAME

dest:SetFit – fit page within window

SYNOPSIS

```
status = dest:SetFit()
```

FUNCTION

`dest:SetFit()` sets the appearance of the page to displaying entire page within the window.

INPUTS

none

RESULTS

status status code

ERRORS

#HPDF_INVALID_DESTINATION - An invalid destination handle was set.

8.2 dest:SetFitB

NAME

dest:SetFitB – fit bounding box of page within window

SYNOPSIS

```
status = dest:SetFitB()
```

FUNCTION

`dest:SetFitB()` sets the appearance of the page to magnifying to fit the bounding box of the page within the window.

INPUTS

none

RESULTS

status status code

ERRORS

#HPDF_INVALID_DESTINATION - An invalid destination handle was set.

8.3 dest:SetFitBH

NAME

dest:SetFitBH – fit bounding box width to window

SYNOPSIS

```
status = dest:SetFitBH(top)
```

FUNCTION

`dest:SetFitBH()` defines the appearance of a page to magnifying to fit the width of the bounding box of the page within the window and setting the top position of the page to the value of the `top` parameter.

INPUTS

`top` the top coordinate of the page

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DESTINATION` - An invalid destination handle was set.

`#HPDF_INVALID_PARAMETER` - An invalid value was set at either left, top or zoom parameter.

8.4 `dest:SetFitBV`

NAME

`dest:SetFitBV` – fit bounding box height to window

SYNOPSIS

`status = dest:SetFitBV(left)`

FUNCTION

`dest:SetFitBV()` defines the appearance of a page to magnifying to fit the height of the bounding box of the page within the window and setting the left position of the page to the value of the `left` parameter.

INPUTS

`left` the left coordinates of the page

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DESTINATION` - An invalid destination handle was set.

`#HPDF_INVALID_PARAMETER` - An invalid value was set at either left, top or zoom parameter.

8.5 `dest:SetFitH`

NAME

`dest:SetFitH` – fit page width to window

SYNOPSIS

`status = dest:SetFitH(top)`

FUNCTION

`dest:SetFitH()` defines the appearance of a page to magnifying to fit the width of the page within the window and setting the top position of the page to the value of the `top` parameter.

INPUTS

`top` the top coordinate of the page

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DESTINATION` - An invalid destination handle was set.

`#HPDF_INVALID_PARAMETER` - An invalid value was set at either left, top or zoom parameter.

8.6 `dest:SetFitR`

NAME

`dest:SetFitR` – fit page to rectangle

SYNOPSIS

```
status = dest:SetFitR(left, bottom, right, top)
```

FUNCTION

`dest:SetFitR()` defines the appearance of a page to magnifying the page to fit a rectangle specified by `left`, `bottom`, `right` and `top`.

INPUTS

`left` the left coordinates of the page

`bottom` the bottom coordinates of the page

`right` the right coordinates of the page

`top` the top coordinates of the page

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DESTINATION` - An invalid destination handle was set.

`#HPDF_INVALID_PARAMETER` - An invalid value was set at either left, top or zoom parameter.

8.7 dest:SetFitV

NAME

dest:SetFitV – fit page height to window

SYNOPSIS

```
status = dest:SetFitV(left)
```

FUNCTION

`dest:SetFitV()` defines the appearance of a page to magnifying to fit the height of the page within the window and setting the left position of the page to the value of the `left` parameter.

INPUTS

`left` the left coordinate of the page

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DESTINATION` - An invalid destination handle was set.

`#HPDF_INVALID_PARAMETER` - An invalid value was set at either `left`, `top` or `zoom` parameter.

8.8 dest:SetXYZ

NAME

dest:SetXYZ – define page appearance

SYNOPSIS

```
status = dest:SetXYZ(left, top, zoom)
```

FUNCTION

`dest:SetXYZ()` defines the appearance of a page with three parameters which are `left`, `top` and `zoom`.

INPUTS

`left` the left coordinates of the page

`top` the top coordinates of the page

`zoom` the page magnified factor; this value must be between 0.08(8%) to 32(3200%)

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DESTINATION` - An invalid destination handle was set.

`#HPDF_INVALID_PARAMETER` - An invalid value was set at either `left`, `top` or `zoom` parameter.

9 Document methods

9.1 doc:AddPage

NAME

doc:AddPage – add new page to document

SYNOPSIS

```
page = doc:AddPage()
```

FUNCTION

`doc:AddPage()` creates a new page and adds it after the last page of a document.

`doc:AddPage()` returns the handle of created page object on success. Otherwise, it returns an error code and the error handler is called.

INPUTS

none

RESULTS

page handle to a page

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

9.2 doc:AddPageLabel

NAME

doc:AddPageLabel – add page labeling range

SYNOPSIS

```
status = doc:AddPageLabel(pagenum, style, firstpage[, prefix])
```

FUNCTION

`doc:AddPageLabel()` adds a page labeling range for the document. The page label is shown in the thumbnails view.

`style` must be one of the following special constants:

#HPDF_PAGE_NUM_STYLE_DECIMAL:
Arabic numerals (1 2 3 4).

#HPDF_PAGE_NUM_STYLE_UPPER_ROMAN:
Uppercase roman numerals (I II III IV).

#HPDF_PAGE_NUM_STYLE_LOWER_ROMAN:
Lowercase roman numerals (i ii iii iv).

#HPDF_PAGE_NUM_STYLE_UPPER_LETTERS:
Uppercase letters (A B C D).

#HPDF_PAGE_NUM_STYLE_LOWER_LETTERS:
 Lowercase letters (a b c d).

When `doc:AddPageLabel()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

pagenum the first page that applies this labeling range

style a valid numbering style (see above)

firstpage
 the first page number to use

prefix optional: the prefix for the page label

RESULTS

status status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_PAGE_NUM_STYLE_OUT_OF_RANGE - An invalid page numbering style is specified.

9.3 doc:AttachFile

NAME

`doc:AttachFile` – attach file to document

SYNOPSIS

`file = doc:AttachFile(f$)`

FUNCTION

`doc:AttachFile()` attaches the file specified by `f$` to the document and returns a handle to the embedded file or `Nil` on error.

INPUTS

f\$ path to a file that should be attached

RESULTS

file handle to the attached file

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

9.4 doc:CreateExtGState

NAME

doc:CreateExtGState – create extended graphics state object

SYNOPSIS

```
egs = doc:CreateExtGState()
```

FUNCTION

`doc:CreateExtGState()` creates a new extended graphics state object.

When `doc:CreateExtGState()` succeeds, it returns the handle of the created extended graphics state object. Otherwise, it returns `Nil` and the error handler is invoked.

INPUTS

none

RESULTS

`egs` handle to an extended graphics state object

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

9.5 doc:CreateImageFromBrush

NAME

doc:CreateImageFromBrush – create new image from Hollywood brush

SYNOPSIS

```
img = doc:CreateImageFromBrush(id[, table])
```

FUNCTION

`doc:CreateImageFromBrush()` creates an image from the Hollywood brush specified by `id`. The image will always use the RGB color space, i.e. `#HPDF_CS_DEVICE_RGB`.

The optional argument `table` can be used to configure further options:

UseJPEG: If this parameter is set to `True`, the image will be compressed using the JPEG file format. You can use the `Quality` field to set the compression level. If `UseJPEG` is set to `False`, the image won't be compressed, but you can use `doc:SetCompressionMode()` to activate compression for image data, although this won't be as good as JPEG. Defaults to `False`.

Quality: Here you can specify a value between 0 and 100 indicating the compression quality for the JPEG format. A value of 100 means best quality, 0 means worst quality. Defaults to 90 which means pretty good quality.

When `doc:CreateImageFromBrush()` succeeds, it returns the handle of an image object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`id` identifier of brush to convert into image

`table` optional: further parameters in a table (see above)

RESULTS

`img` handle to an image

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_INVALID_COLOR_SPACE` - An invalid `color_space` value is specified.

`#HPDF_INVALID_IMAGE` - The size of an image data is invalid.

9.6 doc:CreateImageFromMem

NAME

`doc:CreateImageFromMem` – create new image from memory data

SYNOPSIS

```
img = doc:CreateImageFromMem(data, width, height, colorspace, bpc)
```

FUNCTION

`doc:CreateImageFromMem()` creates an image from raw pixel data in memory. The `data` argument must be a memory pointer obtained via Hollywood's `GetMemPointer()` function. This function loads the data without any conversion so it is usually faster than the other functions. `bpc` specifies the bit size of each color component and can be either 1, 2, 4, or 8.

The `colorspace` argument must be one of `#HPDF_CS_DEVICE_GRAY`, `#HPDF_CS_DEVICE_RGB`, or `#HPDF_CS_DEVICE_CMYK`. See [Section 9.24 \[doc:LoadRawImage\], page 78](#), for details.

When `doc:CreateImageFromMem()` succeeds, it returns the handle of an image object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`data` the pointer to the image data

`width` the width of an image file

`height` the height of an image file

`colorspace`

`#HPDF_CS_DEVICE_GRAY` or `#HPDF_CS_DEVICE_RGB` or `#HPDF_CS_DEVICE_CMYK` is allowed

`bpc` the bit size of each color component; valid values are either 1, 2, 4, 8

RESULTS

`img` handle to an image

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

#HPDF_INVALID_COLOR_SPACE - An invalid color_space value is specified.

#HPDF_INVALID_IMAGE - The size of an image data is invalid.

9.7 doc:CreateOutline

NAME

doc:CreateOutline – create outline object

SYNOPSIS

```
otl = doc:CreateOutline(parent, title, encoder)
```

FUNCTION

`doc:CreateOutline()` creates a new outline object.

When `doc:CreateOutline()` succeeds, it returns the handle of created outline object. Otherwise, it returns `Nil` and the error handler is invoked.

INPUTS

parent the handle of an outline object which comes to the parent of the created outline object; if `Nil`, the outline is created as a root outline

title the caption of the outline object

encoder the handle of an encoding object applied to the title; if `Nil`, the document's encoding is used

RESULTS

otl handle to an outline

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_OUTLINE - An invalid parent outline is specified.

9.8 doc:Free

NAME

doc:Free – free document object

SYNOPSIS

```
doc:Free()
```

FUNCTION

`doc:Free()` frees a document object and all resources.

Note that after calling `doc:Free()` you must no longer use any handles belonging to this document, e.g. page handles, font handles, and of course the document handle itself.

INPUTS

none

9.9 doc:GetCurrentEncoder

NAME

doc:GetCurrentEncoder – get current encoder of document

SYNOPSIS

```
enc = doc:GetCurrentEncoder()
```

FUNCTION

`doc:GetCurrentEncoder()` gets the handle of the current encoder of the document object. The current encoder is set by invoking `doc:SetCurrentEncoder()` and it is used to process text when an application calls `doc:SetInfoAttr()`. The default value of it is `Nil`. It returns a handle of an encoder object or `Nil`.

INPUTS

none

RESULTS

enc handle to an encoder

9.10 doc:GetCurrentPage

NAME

doc:GetCurrentPage – return current page object

SYNOPSIS

```
page = doc:GetCurrentPage()
```

FUNCTION

`doc:GetCurrentPage()` returns the handle of current page object. When `doc:GetCurrentPage()` succeeds, it returns the handle of a current page object. Otherwise it returns `Nil`.

INPUTS

none

RESULTS

page handle to a page

9.11 doc:GetEncoder

NAME

doc:GetEncoder – get encoder object from name

SYNOPSIS

```
enc = doc:GetEncoder(encodingname)
```

FUNCTION

`doc:GetEncoder()` gets the handle of an encoder object by specified encoding name. See [Section 4.11 \[Encodings\]](#), page 14, for a list of valid encoding names.

When `doc:GetEncoder()` succeeds, it returns the handle of an encoder object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`encodingname`
a valid encoding name (see above)

RESULTS

`enc` handle to an encoder

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.
`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.
`#HPDF_INVALID_ENCODING_NAME` - An invalid encoding name was set.

9.12 doc:GetError

NAME

`doc:GetError` – get last error code

SYNOPSIS

```
status = doc:GetError()
```

FUNCTION

`doc:GetError()` returns the last error code of specified document object.

Note that some functions also set a detailed error code. `doc:GetErrorDetail()` can be used to get this detailed error code.

Returns the last error code of document object, or `#HPDF_OK` if no last error.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle is set.

9.13 doc:GetErrorDetail

NAME

`doc:GetErrorDetail` – get detailed error code

SYNOPSIS

```
status = doc:GetErrorDetail()
```

FUNCTION

When an error occurs, some functions set a detailed error code. `doc:GetErrorDetail()` returns this detailed error code.

INPUTS

none

RESULTS

status status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle is set.

9.14 doc:GetFont

NAME

doc:GetFont – get handle of font object

SYNOPSIS

```
font = doc:GetFont(fontname[, encodingname])
```

FUNCTION

`doc:GetFont()` gets the handle of a requested font object.

See [Section 4.6 \[Fonts\]](#), page 12, for a list of valid font names.

See [Section 4.11 \[Encodings\]](#), page 14, for a list of valid encoding names.

When `doc:GetFont()` succeeds, it returns the handle of a font object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

fontname a valid font name

encodingname

optional: a valid encoding name (defaults to current encoding)

RESULTS

font handle to a font

ERRORS

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_INVALID_FONT_NAME - An invalid font name was set.

#HPDF_INVALID_ENCODING_NAME - An invalid encoding name was set.

#HPDF_UNSUPPORTED_FONT_TYPE - An unsupported font type was set.

9.15 doc:GetInfoAttr

NAME

doc:GetInfoAttr – get text from info dictionary

SYNOPSIS

```
str = doc:GetInfoAttr(type)
```

FUNCTION

`doc:GetInfoAttr()` gets an attribute value from info dictionary.

When `doc:GetInfoAttr()` succeeds, it returns the string value of the info dictionary element specified by `type`. If the information has not been set or an error has occurred, it returns `Nil`.

See [Section 9.32 \[doc:SetInfoAttr\]](#), page 83, for possible types that can be passed to this method.

INPUTS

`type` info dictionary element to query

RESULTS

`str` text of info dictionary element

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_INVALID_PARAMETER` - An invalid type parameter was set.

9.16 doc:GetPageByIndex

NAME

`doc:GetPageByIndex` – get page handle from index

SYNOPSIS

```
page = doc:GetPageByIndex(idx)
```

FUNCTION

`doc:GetPageByIndex()` returns the page that is at the specified index.

INPUTS

`idx` page index

RESULTS

`page` handle to a page

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_INVALID_PAGE_INDEX` - The page index is invalid.

9.17 doc:GetPageLayout

NAME

`doc:GetPageLayout` – get current page layout setting

SYNOPSIS

```
layout = doc:GetPageLayout()
```

FUNCTION

`doc:GetPageLayout()` returns the current setting for page layout.

When `doc:GetPageLayout()` succeeds, it returns the current setting for page layout. If page layout is not set, it returns `#HPDF_PAGE_LAYOUT_EOF`.

See [Section 9.35 \[doc:SetPageLayout\]](#), page 85, for possible page layouts.

INPUTS

none

RESULTS

layout page layout constant

9.18 doc:GetPageMode

NAME

`doc:GetPageMode` – get document display mode

SYNOPSIS

```
mode = doc:GetPageMode()
```

FUNCTION

`doc:GetPageMode()` returns the current setting for page mode.

See [Section 9.36 \[doc:SetPageMode\]](#), page 86, for possible page modes.

When `doc:GetPageMode()` succeeds, it returns the current setting for page mode.

INPUTS

none

RESULTS

mode current document page mode

9.19 doc:GetViewerPreference

NAME

`doc:GetViewerPreference` – get viewer preferences

SYNOPSIS

```
flags = doc:GetViewerPreference()
```

FUNCTION

`doc:GetViewerPreference()` gets the viewer preferences for the document.

See [Section 9.40 \[doc:SetViewerPreference\]](#), page 89, for a list of supported preferences.

INPUTS

none

RESULTS

flags viewer preferences for this document

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

9.20 doc:InsertPage**NAME**

doc:InsertPage – insert new page into document

SYNOPSIS

```
page = doc:InsertPage(target)
```

FUNCTION

`doc:InsertPage()` creates a new page and inserts it just before the specified page.

`doc:InsertPage()` returns the handle of the newly created page object on success. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`page` the handle of a page object that should be the successor of the new page

RESULTS

`page` handle to a page

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_PAGE - An invalid page handle was set.

9.21 doc:LoadFont**NAME**

doc:LoadFont – load font using Hollywood

SYNOPSIS

```
font = doc:LoadFont(name[, weight, slant, embed])
```

FUNCTION

`doc:LoadFont()` loads a font using Hollywood and registers it in the document object. If the optional `embed` argument is set to `True`, the glyph data of the font is embedded, otherwise only the matrix data is included in the PDF file.

Note that only TrueType fonts can be used with this method. You cannot use bitmap fonts in PDF documents.

The optional arguments `weight` and `slant` can be used to specify a font weight and slant. The following can be passed in the `weight` parameter:

```
#FONTWEIGHT_THIN
#FONTWEIGHT_EXTRALIGHT
#FONTWEIGHT_ULTRALIGHT
#FONTWEIGHT_LIGHT
```

```

#FONTWEIGHT_BOOK
#FONTWEIGHT_NORMAL (default)
#FONTWEIGHT_REGULAR
#FONTWEIGHT_MEDIUM
#FONTWEIGHT_SEMIBOLD
#FONTWEIGHT_DEMIBOLD
#FONTWEIGHT_BOLD
#FONTWEIGHT_EXTRABOLD
#FONTWEIGHT_ULTRABOLD
#FONTWEIGHT_HEAVY
#FONTWEIGHT_BLACK
#FONTWEIGHT_EXTRABLACK
#FONTWEIGHT_ULTRABLACK

```

The following constants can be passed in the `slant` parameter:

```

#FONTSLANT_ROMAN (default)
#FONTSLANT_ITALIC
#FONTSLANT_OBLIQUE

```

When `doc:LoadTTFont()` succeeds, it returns the name of a font. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

<code>name</code>	name of a font to load through Hollywood
<code>weight</code>	optional: desired font weight (defaults to <code>#FONTWEIGHT_NORMAL</code>)
<code>slant</code>	optional: desired font slant (defaults to <code>#FONTSLANT_ROMAN</code>)
<code>embed</code>	optional: if this parameter is set to <code>True</code> , the glyph data of the font is embedded, otherwise only the matrix data is included in PDF file

RESULTS

<code>font</code>	name of the font as a string
-------------------	------------------------------

ERRORS

```

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.
#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
#HPDF_FONT_EXISTS - The font of the same name has already been registered.
#HPDF_TTF_INVALID_CMAP - Failed to load .ttf file.
#HPDF_TTF_INVALID_FORMAT - Failed to load .ttf file.
#HPDF_TTF_MISSING_TABLE - Failed to load .ttf file.
#HPDF_TTF_CANNOT_EMBEDDING_FONT - The font doesn't allow embedding.

```

9.22 doc:LoadJPEGImage

NAME

`doc:LoadJPEGImage` – load external JPEG image

SYNOPSIS

```
img = doc:LoadJPEGImage(filename)
```

FUNCTION

`doc:LoadJPEGImage()` loads an external JPEG image file.

When `doc:LoadJPEGImage()` succeeds, it returns the handle of an image object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`filename` path to a JPEG image file

RESULTS

`img` handle to an image

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_UNSUPPORTED_JPEG_FORMAT` - Unsupported JPEG image format.

9.23 doc:LoadPNGImage

NAME

`doc:LoadPNGImage` – load external PNG image

SYNOPSIS

```
img = doc:LoadPNGImage(filename[, cache])
```

FUNCTION

`doc:LoadPNGImage()` loads an external PNG image file. The optional `cache` argument allows you to set whether this method should cache the whole PNG image in memory or not. If you need to embed a PNG image several times, it is faster to set this argument to `True`.

Note that when embedding PNG images in a PDF, they are not embedded in PNG format but as raw, uncompressed pixels (although you can activate compression for the pixel data by calling `doc:SetCompressionMode()`). The only image format which can be embedded directly inside PDF documents is JPEG. Use `doc:LoadJPEGImage()` to load a JPEG image for embedding in a PDF.

When `doc:LoadPNGImage()` succeeds, it returns the handle of an image object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`filename` path to a PNG image file

`cache` optional: whether caching should be enabled (defaults to `False`)

RESULTS

`img` handle to an image

ERRORS

- #HPDF_INVALID_DOCUMENT - An invalid document handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
- #HPDF_UNSUPPORTED_FUNC - The library is not configured to use PNGLIB.
- #HPDF_LIBPNG_ERROR - Failed when invoking PNGLIB's function.
- #HPDF_INVALID_PNG_IMAGE - Invalid PNG format.

9.24 doc:LoadRawImage**NAME**

doc:LoadRawImage – load raw image from file

SYNOPSIS

```
img = doc:LoadRawImage(filename, width, height, colorspace)
```

FUNCTION

`doc:LoadRawImage()` loads an image from raw pixel data stored in an external file. This function loads the data without any conversion. So it is usually faster than the other functions. Pixels are stored line by line from top to bottom in the color format specified by the `colorspace` parameter which must be set to one of the following constants:

#HPDF_CS_DEVICE_GRAY:

8 bit gray scale image. The gray scale color space describes each pixel with one byte. For each byte, 0 is maximum dark, and 255 is maximum light. The size of the image data is `width * height` bytes.

#HPDF_CS_DEVICE_RGB:

24 bit RGB color image. The 24 bit RGB color space describes each pixel with three bytes (red, green, blue). For each byte, 0 is maximum dark, 255 maximum light. The size of the image data is `width * height * 3` bytes.

#HPDF_CS_DEVICE_CMYK

32 bit CMYK color image. The 32 bit CMYK color space describes each pixel with four bytes (cyan, magenta, yellow, black). The size of the image data is `width * height * 4` bytes. For each byte, 0 is maximum dark, 255 maximum light.

When `doc:LoadRawImage()` succeeds, it returns the handle of an image object. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

- `filename` a path to an image file
- `width` the width of the raw pixel data
- `height` the height of the raw pixel data
- `colorspace`
 - #HPDF_CS_DEVICE_GRAY, #HPDF_CS_DEVICE_RGB or #HPDF_CS_DEVICE_CMYK (see above)

RESULTS

`img` handle to an image

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_INVALID_COLOR_SPACE` - An invalid `color_space` value is specified.

`#HPDF_INVALID_IMAGE` - The size of an image data is invalid.

`#HPDF_FILE_IO_ERROR` - Cannot read data from the file.

9.25 doc:LoadTTFont**NAME**

`doc:LoadTTFont` – load TrueType font from file

SYNOPSIS

```
font = doc:LoadTTFont(filename, embedding[, index])
```

FUNCTION

`doc:LoadTTFont()` loads a TrueType font from an external file and registers it in the document object. If the optional `index` argument is set to a positive value, this function will load the TrueType font at the specified index from a TrueType collection file instead.

When `doc:LoadTTFont()` succeeds, it returns the name of a font. Otherwise, it returns `Nil` and the error handler is called.

INPUTS

`filename` path to a TrueType font (.ttf) or TrueType font collection (.ttc) file

`embedding`

if this parameter is set to `True`, the glyph data of the font is embedded, otherwise only the matrix data is included in PDF file

`index` optional: index of font to be loaded from TrueType font collection (defaults to -1)

RESULTS

`font` name of the font as a string

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_FONT_EXISTS` - The font of the same name has already been registered.

`#HPDF_INVALID_TTC_INDEX` - The value specified at `index` parameter exceeds the number of fonts.

`#HPDF_INVALID_TTC_FILE` - Failed to load .ttc file.

`#HPDF_TTF_INVALID_CMAP` - Failed to load .ttf file.

`#HPDF_TTF_INVALID_FORMAT` - Failed to load .ttf file.

#HPDF_TTF_MISSING_TABLE - Failed to load .ttf file.
 #HPDF_TTF_CANNOT_EMBEDDING_FONT - The font doesn't allow embedding.

9.26 doc:LoadType1Font

NAME

doc:LoadType1Font – load a Type1 font

SYNOPSIS

```
font = doc:LoadType1Font(afmfilename, pfmfilename)
```

FUNCTION

`doc:LoadType1Font()` loads a Type1 font from an external file and registers it in the document object.

When `doc:LoadType1Font()` succeeds, it returns the name of a font. Otherwise, it returns Nil and the error handler is called.

INPUTS

`afmfilename`
 path to an AFM file

`pfmfilename`
 path to a PFA/PFB file

RESULTS

`font` name of the font as a string

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
 #HPDF_FONT_EXISTS - The font of the same name has already been registered.
 #HPDF_INVALID_AFM_HEADER - Cannot recognize AFM file.
 #HPDF_INVALID_CHAR_MATRICS_DATA - Cannot recognize AFM file.
 #HPDF_INVALID_N_DATA - Cannot recognize AFM file.
 #HPDF_UNSUPPORTED_TYPE1_FONT - Cannot recognize PFA/PFB file.

9.27 doc:ResetError

NAME

doc:ResetError – reset last error code

SYNOPSIS

```
doc:ResetError()
```

FUNCTION

Once an error code is set, IO processing functions cannot be invoked. In the case of executing a function after the cause of the error is fixed, an application have to invoke `doc:ResetError()` to clear error-code before executing functions.

INPUTS

none

9.28 doc:SaveToFile**NAME**

doc:SaveToFile – save document to a file

SYNOPSIS

```
status = doc:SaveToFile(filename)
```

FUNCTION

`doc:SaveToFile()` saves the current document to a file.

Returns `#HPDF_OK` on success, otherwise it returns an error code and the error handler is called.

INPUTS

`filename` The name of file to save.

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle is set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_FILE_IO_ERROR` - An error occurred while processing file I/O.

9.29 doc:SetCompressionMode**NAME**

doc:SetCompressionMode – set document compression mode

SYNOPSIS

```
status = doc:SetCompressionMode(mode)
```

FUNCTION

`doc:SetCompressionMode()` sets the mode of compression. `mode` can be a combination of the following flags:

`#HPDF_COMP_NONE:`

No compression. This cannot be combined with any other flags.

`#HPDF_COMP_TEXT:`

Compress the contents stream of the page.

`#HPDF_COMP_IMAGE:`

Compress the streams of the image objects.

`#HPDF_COMP_METADATA:`

Other stream datas (fonts, cmaps and so on) are compressed.

#HPDF_COMP_ALL:

All stream data is compressed. This is the same as setting `#HPDF_COMP_TEXT`, `#HPDF_COMP_IMAGE`, and `#HPDF_COMP_METADATA` together.

When `doc:SetCompressionMode()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

`mode` a combination of the flags listed above

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_INVALID_COMPRESSION_MODE` - An invalid compression mode was specified.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

9.30 doc:SetCurrentEncoder

NAME

`doc:SetCurrentEncoder` – set current encoder for document

SYNOPSIS

```
status = doc:SetCurrentEncoder(encodingname)
```

FUNCTION

`doc:SetCurrentEncoder()` sets the current encoder for the document.

See [Section 4.11 \[Encodings\]](#), page 14, for a list of valid encoding names.

When `doc:SetCurrentEncoder()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

`encodingname`
the name of an encoding (see above)

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_INVALID_ENCODING_NAME` - An invalid encoding name was set.

9.31 doc:SetEncryptionMode

NAME

doc:SetEncryptionMode – set document encryption mode

SYNOPSIS

```
status = doc:SetEncryptionMode(mode[, keylen])
```

FUNCTION

`doc:SetEncryptionMode()` set the encryption mode. As a side effect, it ups the version of PDF to 1.4 when the mode is set to `#HPDF_ENCRYPT_R3`.

The following encryption modes are currently supported:

`#HPDF_ENCRYPT_R2`:

Use "Revision 2" algorithm. `keylen` is automatically set to 5 (40 bits).

`#HPDF_ENCRYPT_R3`:

Use "Revision 3" algorithm. `keylen` can be 5 (40 bits) to 16 (128bits).

When `doc:SetEncryptionMode()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

`mode` one of the encryption modes listed above

`keylen` specify the byte length of encryption key; only needed for `#HPDF_ENCRYPT_R3` (defaults to 5, i.e. 40 bits)

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_INVALID_ENCRYPT_KEY_LEN` - An invalid key length was specified.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

9.32 doc:SetInfoAttr

NAME

doc:SetInfoAttr – set text of info dictionary attribute

SYNOPSIS

```
status = doc:SetInfoAttr(type, value)
```

FUNCTION

`doc:SetInfoAttr()` sets the text of an info dictionary attribute, using the current encoding of the document. The `type` parameter can be one of the following constants:

`#HPDF_INFO_AUTHOR`:

Document's author

`#HPDF_INFO_CREATOR`:

Document's creator

#HPDF_INFO_TITLE:
Document's title

#HPDF_INFO_SUBJECT:
Document's subject

#HPDF_INFO_KEYWORDS:
Keywords describing the document

When `doc:SetInfoAttr()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

type one of the constants listed above

value text to use for setting the attribute

RESULTS

status status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_PARAMETER - An invalid type parameter was set.

9.33 doc:SetInfoDateAttr

NAME

`doc:SetInfoDateAttr` – set a datetime attribute in info dictionary

SYNOPSIS

`status = doc:SetInfoDateAttr(type, value)`

FUNCTION

`doc:SetInfoDateAttr()` sets a datetime attribute in the info dictionary. `type` must be one of the following constants:

#HPDF_INFO_CREATION_DATE:
Document's creation date

#HPDF_INFO_MOD_DATE:
Document's last modification date

`value` must be a table containing a datetime description. The table must contain the following fields:

Day: Between 1 and 31 (depends on the month).

Month: Between 1 and 12.

Year: The year.

Hour: Between 0 and 23.

Minutes: Between 0 and 59.

Seconds: Between 0 and 59.

Ind: Relationship of local time to Universal Time. This can be " ", "+", "-", or "Z".

Off_Hour:
If **ind** is not space, 0 to 23 is valid. Otherwise, ignored.

Off_Minutes:
If **ind** is not space, 0 to 59 is valid. Otherwise, ignored.

INPUTS

type one of the constants listed above

value table containing a datetime description

RESULTS

status status code

9.34 doc:SetOpenAction**NAME**

`doc:SetOpenAction` – set document's initial page

SYNOPSIS

```
status = doc:SetOpenAction(dst)
```

FUNCTION

`doc:SetOpenAction()` set the first page to appear when a document is opened. **dst** must be a valid destination object created by `page>CreateDestination()`.

When `doc:SetOpenAction()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

dst valid destination object

RESULTS

status status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_INVALID_DESTINATION` - An invalid destination object was set.

9.35 doc:SetPageLayout**NAME**

`doc:SetPageLayout` – set how pages should be displayed

SYNOPSIS

```
status = doc:SetPageLayout(layout)
```

FUNCTION

`doc:SetPageLayout()` sets how the pages should be displayed. If this attribute is not set, the setting of the viewer application is used.

`layout` can be one of the following constants:

#HPDF_PAGE_LAYOUT_SINGLE:

Only one page is displayed.

#HPDF_PAGE_LAYOUT_ONE_COLUMN:

Display the pages in one column.

#HPDF_PAGE_LAYOUT_TWO_COLUMN_LEFT:

Display in two columns. Odd page number is displayed left.

#HPDF_PAGE_LAYOUT_TWO_COLUMN_RIGHT:

Display in two columns. Odd page number is displayed right.

When `doc:SetPageLayout()` succeeds, it returns **#HPDF_OK**. Otherwise, it returns an error code and the error handler is called.

INPUTS

`layout` one of the page layout constants (see above)

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle is set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_PAGE_LAYOUT_OUT_OF_RANGE - An invalid page layout is specified.

9.36 doc:SetPageMode

NAME

`doc:SetPageMode` – set how document should be displayed

SYNOPSIS

```
status = doc:SetPageMode(mode)
```

FUNCTION

`doc:SetPageMode()` sets how the document should be displayed.

`mode` can be one of the following constants:

#HPDF_PAGE_MODE_USE_NONE:

Display the document with neither outline nor thumbnail.

#HPDF_PAGE_MODE_USE_OUTLINE:

Display the document with outline pane.

#HPDF_PAGE_MODE_USE_THUMBS:
 Display the document with thumbnail pane.

#HPDF_PAGE_MODE_FULL_SCREEN:
 Display the document with full screen mode.

When `doc:SetPageMode()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

`mode` a valid page mode (see above for possible options)

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle is set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_PAGE_MODE_OUT_OF_RANGE - An invalid page mode is specified.

9.37 doc:SetPagesConfiguration

NAME

`doc:SetPagesConfiguration` – set maximum number of pages

SYNOPSIS

`status = doc:SetPagesConfiguration(page_per_pages)`

FUNCTION

In the default setting, a document object has one "Pages" object as root of pages. All "Page" objects are created as children of the "Pages" object. Since a "Pages" object can own only 8191 children objects, the maximum number of pages are 8191 pages. Additionally, the state that there are a lot of "Page" object under one "Pages" object is not good, because it causes performance degradation of a viewer application.

An application can change the setting of a pages tree by invoking `doc:SetPagesConfiguration()`. If `page_per_pages` parameter is set to more than zero, a two-tier pages tree is created. A root "Pages" object can own 8191 "Pages" object, and each lower "Pages" object can own `page_per_pages` "Page" objects. As a result, the maximum number of pages becomes $8191 * \text{page_per_pages}$ page. An application cannot invoke `doc:SetPagesConfiguration()` after a page is added to document.

When `doc:SetPagesConfiguration()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

`page_per_pages`
 specify the numbers of pages that a "Pages" object can own.

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle is set.

#HPDF_INVALID_DOCUMENT_STATE - A page object already exists in a document.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

9.38 doc:SetPassword**NAME**

doc:SetPassword – set document password

SYNOPSIS

```
status = doc:SetPassword(ownerpwd[, userpwd])
```

FUNCTION

`doc:SetPassword()` sets a password for the document. If the password is set, document contents are encrypted. The owner can change the permission of the document. Note that the owner password must not be the same as the user password. The user password is optional.

When `doc:SetPassword()` succeeds, it returns #HPDF_OK. Otherwise, it returns an error code and the error handler is called.

INPUTS

`ownerpwd` the password for the owner of the document

`userpwd` optional: the password for the user of the document.

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_PASSWORD - Owner password is Nil, zero length string, or same value as user password.

9.39 doc:SetPermission**NAME**

doc:SetPermission – set document permissions

SYNOPSIS

```
status = doc:SetPermission(permission)
```

FUNCTION

`doc:SetPermission()` sets the permission flags for the document. `permission` must be combination of the following flags:

#HPDF_ENABLE_READ:

User can read the document.

```
#HPDF_ENABLE_PRINT:
    User can print the document.

#HPDF_ENABLE_EDIT_ALL:
    User can edit the contents of the document other than annotations, form
    fields.

#HPDF_ENABLE_COPY:
    User can copy the text and the graphics of the document.

#HPDF_ENABLE_EDIT:
    User can add or modify the annotations and form fields of the document.
```

When `doc:SetPermission()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is called.

INPUTS

```
permission
    one or more permission flags (see above)
```

RESULTS

```
status    status code
```

ERRORS

```
#HPDF_INVALID_DOCUMENT - An invalid document handle was set.
#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
```

9.40 doc:SetViewerPreference

NAME

`doc:SetViewerPreference` – set viewer preferences

SYNOPSIS

```
status = doc:SetViewerPreference(flags)
```

FUNCTION

`doc:SetViewerPreference()` sets the viewer preferences for the document.

`flags` can be a combination of the following options:

```
#HPDF_HIDE_TOOLBAR:
    Hide viewer's toolbar.

#HPDF_HIDE_MENUBAR:
    Hide viewer's menu bar.

#HPDF_HIDE_WINDOW_UI
    Hide viewer's user interface.

#HPDF_FIT_WINDOW:
    Fit document in viewer window.

#HPDF_CENTER_WINDOW:
    Center document in viewer window.
```

#HPDF_PRINT_SCALING_NONE:
 Disable scaling when printing.

INPUTS

flags one or more viewer flags (see above)

RESULTS

status status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

9.41 doc:UseCNSEncodings

NAME

`doc:UseCNSEncodings` – enable simplified Chinese encodings

SYNOPSIS

```
status = doc:UseCNSEncodings()
```

FUNCTION

`doc:UseCNSEncodings()` enables simplified Chinese encodings. After `doc:UseCNSEncodings()` is invoked, an application can use the following simplified Chinese encodings:

- GB-EUC-H
- GB-EUC-V
- GBK-EUC-H
- GBK-EUC-V

When `doc:UseCNSEncodings()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

status status code

ERRORS

#HPDF_INVALID_DOCUMENT - An invalid document handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_DUPLICATE_REGISTRATION - The encoding of the same name has already been registered.

9.42 doc:UseCNSFonts

NAME

doc:UseCNSFonts – enable simplified Chinese fonts

SYNOPSIS

```
status = doc:UseCNSFonts()
```

FUNCTION

`doc:UseCNSFonts()` enables simplified Chinese fonts. After `doc:UseCNSFonts()` has been called, an application can use the following simplified Chinese fonts:

- SimSun
- SimSun,Bold
- SimSun,Italic
- SimSun,BoldItalic
- SimHei
- SimHei,Bold
- SimHei,Italic
- SimHei,BoldItalic

When `doc:UseCNSFonts()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

```
status    status code
```

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The font of the same name has already been registered.

9.43 doc:UseCNTEncodings

NAME

doc:UseCNTEncodings – enable traditional Chinese encodings

SYNOPSIS

```
status = doc:UseCNTEncodings()
```

FUNCTION

`doc:UseCNTEncodings()` enables traditional Chinese encodings. After `doc:UseCNTEncodings()` is invoked, an application can use the following traditional Chinese encodings:

- GB-EUC-H

- GB-EUC-V
- GBK-EUC-H
- GBK-EUC-V

When `doc:UseCNTEncodings()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The encoding of the same name has already been registered.

9.44 doc:UseCNTFonts

NAME

`doc:UseCNTFonts` – enable traditional Chinese fonts

SYNOPSIS

```
status = doc:UseCNTFonts()
```

FUNCTION

`doc:UseCNTFonts()` enables traditional Chinese fonts. After `doc:UseCNTFonts()` has been called, an application can use the following traditional Chinese fonts:

- MingLiU
- MingLiU,Bold
- MingLiU,Italic
- MingLiU,BoldItalic

When `doc:UseCNSFonts()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The font of the same name has already been registered.

9.45 doc:UseJPEncodings

NAME

doc:UseJPEncodings – enable Japanese encodings

SYNOPSIS

```
status = doc:UseJPEncodings()
```

FUNCTION

`doc:UseJPEncodings()` enables Japanese encodings. After `doc:UseJPEncodings()` is invoked, an application can use the following Japanese encodings:

- 90ms-RKSJ-H
- 90ms-RKSJ-V
- 90msp-RKSJ-H
- EUC-H
- EUC-V

When `doc:UseJPEncodings()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The encoding of the same name has already been registered.

9.46 doc:UseJPFonts

NAME

doc:UseJPFonts – enable Japanese fonts

SYNOPSIS

```
status = doc:UseJPFonts()
```

FUNCTION

`doc:UseJPFonts()` enables Japanese fonts. After `doc:UseJPFonts()` has been called, an application can use the following Japanese fonts:

- MS-Mincyo
- MS-Mincyo,Bold
- MS-Mincyo,Italic
- MS-Mincyo,BoldItalic
- MS-Gothic

- MS-Gothic,Bold
- MS-Gothic,Italic
- MS-Gothic,BoldItalic
- MS-PMincyo
- MS-PMincyo,Bold
- MS-PMincyo,Italic
- MS-PMincyo,BoldItalic
- MS-PGothic
- MS-PGothic,Bold
- MS-PGothic,Italic
- MS-PGothic,BoldItalic

When `doc:UseJPFonts()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The font of the same name has already been registered.

9.47 doc:UseKREncodings

NAME

`doc:UseKREncodings` – enable Korean encodings

SYNOPSIS

```
status = doc:UseKREncodings()
```

FUNCTION

`doc:UseKREncodings()` enables Korean encodings. After `doc:UseKREncodings()` is invoked, an application can use the following Korean encodings:

- KSC-EUC-H
- KSC-EUC-V
- KSCms-UHC-H
- KSCms-UHC-HW-H
- KSCms-UHC-HW-V

When `doc:UseKREncodings()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The encoding of the same name has already been registered.

9.48 `doc:UseKRFonts`

NAME

`doc:UseKRFonts` – enable Korean fonts

SYNOPSIS

```
status = doc:UseKRFonts()
```

FUNCTION

`doc:UseKRFonts()` enables Korean fonts. After `doc:UseKRFonts()` has been called, an application can use the following Korean fonts:

- DotumChe
- DotumChe,Bold
- DotumChe,Italic
- DotumChe,BoldItalic
- Dotum
- Dotum,Bold
- Dotum,Italic
- Dotum,BoldItalic
- BatangChe
- BatangChe,Bold
- BatangChe,Italic
- BatangChe,BoldItalic
- Batang
- Batang,Bold
- Batang,Italic
- Batang,BoldItalic

When `doc:UseKRFonts()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The font of the same name has already been registered.

9.49 doc:UseUTFEncodings**NAME**

`doc:UseUTFEncodings` – enable UTF-8 encodings

SYNOPSIS

`status = doc:UseUTFEncodings()`

FUNCTION

`doc:UseUTFEncodings()` enables UTF-8 encodings. After `doc:UseUTFEncodings()` is invoked, an application can include UTF-8 encoded Unicode text (up to 3-byte UTF-8 sequences only). An application can use the following Unicode encodings (but only with TrueType fonts):

- UTF-8

When `doc:UseUTFEncodings()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_DOCUMENT` - An invalid document handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_DUPLICATE_REGISTRATION` - The encoding of the same name has already been registered.

10 Encoder methods

10.1 encoder:GetByteType

NAME

encoder:GetByteType – byte type in text

SYNOPSIS

```
t = encoder:GetByteType(text, index)
```

FUNCTION

`encoder:GetByteType()` returns the type of byte in the text at the specified position index.

INPUTS

<code>text</code>	text string
<code>index</code>	index within the text string

RESULTS

<code>t</code>	byte type
----------------	-----------

10.2 encoder:GetType

NAME

encoder:GetType – get type of encoding object

SYNOPSIS

```
t = encoder:GetType()
```

FUNCTION

`encoder:GetType()` gets the type of an encoding object.

INPUTS

none

RESULTS

<code>t</code>	encoder type
----------------	--------------

10.3 encoder:GetUnicode

NAME

encoder:GetUnicode – convert character to Unicode

SYNOPSIS

```
unicode = encoder:GetUnicode(code)
```

FUNCTION

`encoder:GetUnicode()` converts a specified character code to Unicode.

INPUTS

`code` a character code to convert

RESULTS

`ucode` character code in Unicode

10.4 `encoder:GetWritingMode`

NAME

`encoder:GetWritingMode` – get writing mode of encoding object

SYNOPSIS

```
mode = encoder:GetWritingMode()
```

FUNCTION

`encoder:GetWritingMode()` returns the writing mode for the encoding object.

INPUTS

none

RESULTS

`mode` writing mode

11 ExtGState methods

11.1 extgs:SetAlphaFill

NAME

extgs:SetAlphaFill – set filling transparency

SYNOPSIS

```
status = extgs:SetAlphaFill(value)
```

FUNCTION

`extgs:SetAlphaFill()` defines the transparency for filling.

When `extgs:SetAlphaFill()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

value the alpha value for filling; it must be between 0 and 1

RESULTS

status status code

ERRORS

`#HPDF_INVALID_OBJECT` - An invalid ExtGState handle was set.

`#HPDF_EXT_GSTATE_READ_ONLY` - The ExtGState object is read only.

`#HPDF_EXT_GSTATE_OUT_OF_RANGE` - An invalid value was set at value parameter.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

11.2 extgs:SetAlphaStroke

NAME

extgs:SetAlphaStroke – set stroking transparency

SYNOPSIS

```
status = extgs:SetAlphaStroke(value)
```

FUNCTION

`extgs:SetAlphaStroke()` defines the transparency for stroking.

When `extgs:SetAlphaStroke()` succeeds, it returns `#HPDF_OK`. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

value the alpha value for stroking; it must be between 0 and 1

RESULTS

status status code

ERRORS

`#HPDF_INVALID_OBJECT` - An invalid ExtGState handle was set.

#HPDF_EXT_GSTATE_READ_ONLY - The ExtGState object is read only.
 #HPDF_EXT_GSTATE_OUT_OF_RANGE - An invalid value was set at value parameter.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

11.3 extgs:SetBlendMode

NAME

extgs:SetBlendMode – set blend mode

SYNOPSIS

```
status = extgs:SetBlendMode(bmode)
```

FUNCTION

`extgs:SetBlendMode()` sets the method of blending.

The `bmode` parameter must be one of the following constants:

```
#HPDF_BM_NORMAL
#HPDF_BM_MULTIPLY
#HPDF_BM_SCREEN
#HPDF_BM_OVERLAY
#HPDF_BM_DARKEN
#HPDF_BM_LIGHTEN
#HPDF_BM_COLOR_DODGE
#HPDF_BM_COLOR_BUM
#HPDF_BM_HARD_LIGHT
#HPDF_BM_SOFT_LIGHT
#HPDF_BM_DIFFERENCE
#HPDF_BM_EXCLUSHON
```

When `extgs:SetBlendMode()` succeeds, it returns #HPDF_OK. Otherwise, it returns an error code and the error handler is invoked.

INPUTS

`bmode` desired blend mode (see above for possible values)

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_OBJECT - An invalid ExtGState handle was set.
 #HPDF_EXT_GSTATE_READ_ONLY - The ExtGState object is read only.
 #HPDF_EXT_GSTATE_OUT_OF_RANGE - An invalid value was set at value parameter.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

12 Font methods

12.1 font:GetAscent

NAME

font:GetAscent – get vertical ascent of font

SYNOPSIS

```
asc = font:GetAscent()
```

FUNCTION

`font:GetAscent()` gets the vertical ascent of the font.

Returns font vertical ascent on success. Otherwise, returns 0.

INPUTS

none

RESULTS

`asc` vertical ascent

12.2 font:GetBBox

NAME

font:GetBBox – get font bounding box

SYNOPSIS

```
bbox = font:GetBBox()
```

FUNCTION

`font:GetBBox()` gets the bounding box of the font. This returns a table that has the `left`, `top`, `right`, and `bottom` fields initialized. On success, the fields are set to the font's bounding box, otherwise all fields are 0.

INPUTS

none

RESULTS

`bbox` font's bounding box

12.3 font:GetCapHeight

NAME

font:GetCapHeight – get uppercase baseline distance

SYNOPSIS

```
ch = font:GetCapHeight()
```

FUNCTION

`font:GetCapHeight()` gets the distance from the baseline of uppercase letters.

Returns font cap height on success. Otherwise, returns 0.

INPUTS

none

RESULTS

ch font cap height

12.4 font:GetDescent

NAME

font:GetDescent – get vertical descent of font

SYNOPSIS

```
desc = font:GetDescent()
```

FUNCTION

font:GetDescent() gets the vertical descent of the font.

Returns font vertical descent on success. Otherwise, returns 0.

INPUTS

none

RESULTS

desc vertical descent

12.5 font:GetEncodingName

NAME

font:GetEncodingName – get font's encoding name

SYNOPSIS

```
name = font:GetEncodingName()
```

FUNCTION

font:GetEncodingName() gets the encoding name of the font.

Returns font encoding name on success. Otherwise, returns Nil.

INPUTS

none

RESULTS

name font encoding name

12.6 font:GetFontName

NAME

font:GetFontName – get font name

SYNOPSIS

```
name = font:GetFontName()
```

FUNCTION

`font:GetFontName()` gets the name of the font.

Returns font name on success. Otherwise, returns `Nil`.

INPUTS

none

RESULTS

`name` font name

12.7 font:GetUnicodeWidth

NAME

`font:GetUnicodeWidth` – get Unicode character width

SYNOPSIS

```
w = font:GetUnicodeWidth(code)
```

FUNCTION

`font:GetUnicodeWidth()` gets the width of a Unicode character in a specific font. The actual width of the character on the page can be calculated as follows:

```
char_width = font:GetUnicodeWidth(font, UNICODE)
actual_width = char_width * FONT_SIZE / 1000
```

Returns character width on success. Otherwise, returns `Nil`.

INPUTS

`code` a Unicode character

RESULTS

`w` Unicode character width

12.8 font:GetXHeight

NAME

`font:GetXHeight` – get lowercase baseline distance

SYNOPSIS

```
xh = font:GetXHeight()
```

FUNCTION

`font:GetXHeight()` gets the distance from the baseline of lowercase letters.

Returns font x-height value on success. Otherwise, returns 0.

INPUTS

none

RESULTS

`xh` x height value

12.9 font:MeasureText

NAME

font:MeasureText – calculate text byte length

SYNOPSIS

```
bl, rw = font:MeasureText(text, len, width, fontsize, charspace,
                          wordspace, wordwrap)
```

FUNCTION

font:MeasureText() calculates the byte length which can be included within the specified width.

The **wordwrap** parameter configures how words should be wrapped: Suppose there are three words: "ABCDE", "FGH", and "IJKL". Also, suppose the substring until "J" can be included within the width (12 bytes). If **wordwrap** is **False** the function returns 12. If **wordwrap** parameter is **True**, it returns 10 (the end of the previous word).

On success, returns byte length which can be included within specified width. Otherwise, returns 0.

INPUTS

text the text to use for calculation

len the length of the text

width the width of the area to put the text

fontsize the size of the font

charspace the character spacing

wordspace the word spacing

wordwrap boolean indicating whether to enable wordwrapping

RESULTS

bl byte length

rw real width of text

12.10 font:TextWidth

NAME

font:TextWidth – get text width

SYNOPSIS

```
t = font:TextWidth(text, len)
```

FUNCTION

font:TextWidth() gets the total width of the text, the number of characters, and the number of words.

This method returns a table that has the following fields initialized:

NumChars:

The number of characters.

NumWords:

The number of words (obsolete). Use **NumSpace** instead (see below).

Width: The total width of the text.

NumSpace:

The number of words.

In case of an error, all table elements will be set to 0.

INPUTS

text the text to get width

len the byte length of the text

RESULTS

t table containing calculation results

13 Image methods

13.1 image:AddSMask

NAME

image:AddSMask – add stencil mask

SYNOPSIS

```
status = image:AddSMask(smask)
```

FUNCTION

`image:AddSMask()` adds a stencil mask image. `smask` must be a gray-scale image.

INPUTS

`smask` handle of an image object which is used as the stencil mask

RESULTS

`status` status code

13.2 image:GetBitsPerComponent

NAME

image:GetBitsPerComponent – get bits per component

SYNOPSIS

```
bpc = image:GetBitsPerComponent()
```

FUNCTION

`image:GetBitsPerComponent()` gets the number of bits used to describe each color component.

INPUTS

none

RESULTS

`bpc` bits per component

13.3 image:GetColorSpace()

NAME

image:GetColorSpace() – get image color space

SYNOPSIS

```
name = image:GetColorSpace()
```

FUNCTION

`image:GetColorSpace()` gets the name of the image's color space.

INPUTS

none

RESULTS

name color space name

13.4 image:GetHeight

NAME

image:GetHeight – get image height

SYNOPSIS

```
h = image:GetHeight()
```

FUNCTION

`image:GetHeight()` gets the height of the image of an image object.

INPUTS

none

RESULTS

h image height

13.5 image:GetSize

NAME

image:GetSize – get image size

SYNOPSIS

```
w,h = image:GetSize()
```

FUNCTION

`image:GetSize()` gets the size of the image of an image object.

INPUTS

none

RESULTS

w image width

h image height

13.6 image:GetWidth

NAME

image:GetWidth – get image width

SYNOPSIS

```
w = image:GetWidth()
```

FUNCTION

`image:GetWidth()` gets the width of the image of an image object.

INPUTS

none

RESULTS

w image width

13.7 image:SetColorMask

NAME

image:SetColorMask – set transparent color

SYNOPSIS

status = image:SetColorMask(rmin, rmax, gmin, gmax, bmin, bmax)

FUNCTION

`image:SetColorMask()` sets the transparent color of the image by the RGB range values. The color within the range is displayed as a transparent color. The image must be in RGB color space.

INPUTS

rmin the lower limit of red; it must be between 0 and 255
 rmax the upper limit of red; it must be between 0 and 255
 gmin the lower limit of green; it must be between 0 and 255
 gmax the upper limit of green; it must be between 0 and 255
 bmin the lower limit of blue; it must be between 0 and 255
 bmax the upper limit of blue; it must be between 0 and 255

RESULTS

status status code

ERRORS

#HPDF_INVALID_IMAGE - An invalid image handle was set.
 #HPDF_INVALID_COLOR_SPACE - An image other than RGB color was specified.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
 #HPDF_INVALID_PARAMETER - An invalid value is specified.

13.8 image:SetMaskImage

NAME

image:SetMaskImage – set mask image

SYNOPSIS

status = image:SetMaskImage(maskimage)

FUNCTION

`image:SetMaskImage()` sets the mask image. `maskimage` must be a 1-bit gray-scale image.

INPUTS

`maskimage` handle of an image object which is used as the image mask

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_IMAGE` - An invalid image handle was set.

`#HPDF_INVALID_BIT_PER_COMPONENT` - An invalid bit-per-component.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

14 Outline methods

14.1 outline:SetDestination

NAME

outline:SetDestination – set destination object

SYNOPSIS

```
status = outline:SetDestination(dst)
```

FUNCTION

`outline:SetDestination()` sets a destination object which becomes a target to jump to when the outline is clicked.

INPUTS

`dst` specify the handle of an destination object

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_OUTLINE - An invalid outline handle was set.

#HPDF_INVALID_DESTINATION - An invalid destination handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

14.2 outline:SetOpened

NAME

outline:SetOpened – set node's open mode

SYNOPSIS

```
status = outline:SetOpened(opened)
```

FUNCTION

`outline:SetOpened()` sets whether this node is opened or not when the outline is displayed for the first time.

INPUTS

`opened` specify whether the node is opened or not

RESULTS

`status` status code

ERRORS

#HPDF_INVALID_OUTLINE - An invalid outline handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

15 Page methods

15.1 page:Arc

NAME

page:Arc – append arc to path

SYNOPSIS

```
status = page:Arc(x, y, radius, ang1, ang2)
```

FUNCTION

`page:Arc()` appends a circle arc to the current path. Angles are given in degrees, with 0 degrees being vertical, upward, from the (x,y) position.

INPUTS

x	the x center point of the circle
y	the y center point of the circle
radius	the radius of the circle
ang1	the angle of the beginning of the arc
ang2	the angle of the end of the arc; it must be greater than ang1

RESULTS

status	status code
--------	-------------

15.2 page:BeginText

NAME

page:BeginText – begin text object

SYNOPSIS

```
status = page:BeginText()
```

FUNCTION

`page:BeginText()` begins a text object and sets the text position to (0, 0).

INPUTS

none

RESULTS

status	status code
--------	-------------

15.3 page:Circle

NAME

page:Circle – append circle to path

SYNOPSIS

```
status = page:Circle(x, y, radius)
```

FUNCTION

page:Circle() appends a circle to the current path.

INPUTS

x x center point of the circle
y y center point of the circle
radius the radius of the circle

RESULTS

status status code

15.4 page:Clip

NAME

page:Clip – modify clipping path

SYNOPSIS

```
status = page:Clip()
```

FUNCTION

page:Clip() modifies the current clipping path by intersecting it with the current path using the nonzero winding number rule. The clipping path is only modified after the succeeding painting operator. To avoid painting the current path, use the function page:EndPath().

Following painting operations will only affect the regions of the page contained by the clipping path. Initially, the clipping path includes the entire page. There is no way to enlarge the current clipping path, or to replace the clipping path with a new one. The functions page:GSave() and page:GRestore() may be used to save and restore the current graphics state, including the clipping path.

INPUTS

none

RESULTS

status status code

15.5 page:ClosePath

NAME

page:ClosePath – close subpath

SYNOPSIS

```
status = page:ClosePath()
```

FUNCTION

`page:ClosePath()` appends a straight line from the current point to the start point of sub path. The current point is moved to the start point of sub path.

INPUTS

none

RESULTS

`status` status code

15.6 page:ClosePathEofillStroke

NAME

page:ClosePathEofillStroke – close, even odd fill and paint path

SYNOPSIS

```
status = page:ClosePathEofillStroke()
```

FUNCTION

`page:ClosePathEofillStroke()` closes the current path, fills the current path using the even-odd rule, then paints the path.

INPUTS

none

RESULTS

`status` status code

15.7 page:ClosePathFillStroke

NAME

page:ClosePathFillStroke – close, winding fill and paint path

SYNOPSIS

```
status = page:ClosePathFillStroke()
```

FUNCTION

`page:ClosePathFillStroke()` closes the current path, fills the current path using the nonzero winding number rule, then paints the path.

INPUTS

none

RESULTS

`status` status code

15.8 page:ClosePathStroke**NAME**

`page:ClosePathStroke` – close and paint path

SYNOPSIS

`status = page:ClosePathStroke()`

FUNCTION

`page:ClosePathStroke()` closes the current path. Then it paints the path.

INPUTS

none

RESULTS

`status` status code

15.9 page:Concat**NAME**

`page:Concat` – concatenate matrix

SYNOPSIS

`status = page:Concat(a, b, c, d, x, y)`

FUNCTION

`page:Concat()` concatenates the page's current transformation matrix and the specified matrix.

For example, if you want to rotate the coordinate system of the page by 45 degrees, use `page:Concat()` as follows:

```
Local rad1 = 45 / 180 * #PI
page:Concat(Cos(rad1),Sin(rad1),-Sin(rad1),Cos(rad1),220,350)
```

To change the coordinate system of the page to 300 dpi, use `page:Concat()` as follows:

```
page:Concat(72.0 / 300.0, 0, 0, 72.0 / 300.0, 0, 0)
```

Invoke `page:GSave()` before `page:Concat()`. Then the change by `page:Concat()` can be restored by invoking `page:GRestore()`.

```
; save the current graphics states
page:GSave(page)

; concatenate the transformation matrix
page:Concat(72.0 / 300.0, 0, 0, 72.0 / 300.0, 0, 0)

; show text on the translated coordinates
```

```

page:BeginText()
page:MoveTextPos(50, 100)
page:ShowText("Text on the translated coordinates")
page:EndText(page)

; restore the graphics states
page:GRestore()

```

An application can invoke `page:GSave()` when the graphics mode of the page is in `#HPDF_GMODE_PAGE_DESCRIPTION`.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

a	scaling x coordinate
b	rotation x coordinate
c	rotation y coordinate
d	scaling y coordinate
x	translation x coordinate
y	translation y coordinate

RESULTS

status	status code
--------	-------------

15.10 page:CreateCircleAnnot

NAME

`page:CreateCircleAnnot` – create circle annotation object

SYNOPSIS

```
ant = page:CreateCircleAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateCircleAnnot()` creates a new circle annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

<code>rect</code>	a rectangle of the clickable area
<code>text</code>	the text to be displayed
<code>encoder</code>	an encoder handle which is used to encode the text; if it is <code>Nil</code> , the default encoding is used

RESULTS

<code>ant</code>	handle to an annotation
------------------	-------------------------

ERRORS

- #HPDF_INVALID_PAGE - An invalid page handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
- #HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.11 page:CreateDestination**NAME**

page:CreateDestination – create destination object

SYNOPSIS

```
dst = page:CreateDestination()
```

FUNCTION

`page:CreateDestination()` creates a new destination object for the page.

INPUTS

none

RESULTS

`dst` handle to a destination

ERRORS

- #HPDF_INVALID_PAGE - An invalid page handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

15.12 page:CreateFreeTextAnnot**NAME**

page:CreateFreeTextAnnot – create free text annotation object

SYNOPSIS

```
ant = page:CreateFreeTextAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateFreeTextAnnot()` creates a new free text annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

- `rect` a rectangle of the clickable area
- `text` the text to be displayed
- `encoder` an encoder handle which is used to encode the text; if it is `Nil`, the default encoding is used

RESULTS

`ant` handle to an annotation

ERRORS

- #HPDF_INVALID_PAGE - An invalid page handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
- #HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.13 page:CreateHighlightAnnot**NAME**

page:CreateHighlightAnnot – create highlight annotation object

SYNOPSIS

```
ant = page:CreateHighlightAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateHighlightAnnot()` creates a new highlight annotation object for the page. The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

- `rect` a rectangle of the clickable area
- `text` the text to be displayed
- `encoder` an encoder handle which is used to encode the text; if it is `Nil`, the default encoding is used

RESULTS

- `ant` handle to an annotation

ERRORS

- #HPDF_INVALID_PAGE - An invalid page handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
- #HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.14 page:CreateLineAnnot**NAME**

page:CreateLineAnnot – create line annotation object

SYNOPSIS

```
ant = page:CreateLineAnnot(text, encoder)
```

FUNCTION

`page:CreateLineAnnot()` creates a new line annotation object for the page.

INPUTS

- `text` the text to be displayed
- `encoder` an encoder handle which is used to encode the text; if it is `Nil`, the default encoding is used

RESULTS

`ant` handle to an annotation

ERRORS

#HPDF_INVALID_PAGE - An invalid page handle was set.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
 #HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.15 page:CreateLinkAnnot**NAME**

`page:CreateLinkAnnot` – create link annotation object

SYNOPSIS

`ant = page:CreateLinkAnnot(rect, dst)`

FUNCTION

`page:CreateLinkAnnot()` creates a new link annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

`rect` a rectangle of clickable area
`dst` a handle of destination object to jump to

RESULTS

`ant` handle to an annotation

ERRORS

#HPDF_INVALID_PAGE - An invalid page handle was set.
 #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
 #HPDF_INVALID_DESTINATION - An invalid destination handle is specified.

15.16 page:CreatePopupAnnot**NAME**

`page:CreatePopupAnnot` – create popup annotation object

SYNOPSIS

`ant = page:CreatePopupAnnot(rect, parent)`

FUNCTION

`page:CreatePopupAnnot()` creates a new popup annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

`rect` a rectangle of the clickable area

`parent` parent annotation object

RESULTS

`ant` handle to an annotation

15.17 `page:CreateProjectionAnnot`

NAME

`page:CreateProjectionAnnot` – create projection annotation object

SYNOPSIS

```
ant = page:CreateProjectionAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateProjectionAnnot()` creates a new projection annotation object for the page. The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

`rect` a rectangle of the clickable area

`text` the text to be displayed

`encoder` an encoder handle which is used to encode the text; if it is `Nil`, the default encoding is used

RESULTS

`ant` handle to an annotation

ERRORS

`#HPDF_INVALID_PAGE` - An invalid page handle was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

`#HPDF_INVALID_ENCODER` - An invalid encoder handle is specified.

15.18 `page:CreateSquareAnnot`

NAME

`page:CreateSquareAnnot` – create square annotation object

SYNOPSIS

```
ant = page:CreateSquareAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateSquareAnnot()` creates a new square annotation object for the page. The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

`rect` a rectangle of the clickable area

text the text to be displayed

encoder an encoder handle which is used to encode the text; if it is Nil, the default encoding is used

RESULTS

ant handle to an annotation

ERRORS

#HPDF_INVALID_PAGE - An invalid page handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.19 page:CreateSquigglyAnnot**NAME**

page:CreateSquigglyAnnot – create squiggly annotation object

SYNOPSIS

ant = page:CreateSquigglyAnnot(**rect**, **text**, **encoder**)

FUNCTION

page:CreateSquigglyAnnot() creates a new squiggly annotation object for the page. The **rect** parameter must be a table which contains **left**, **top**, **right**, and **bottom** fields that describe a rectangle.

INPUTS

rect a rectangle of the clickable area

text the text to be displayed

encoder an encoder handle which is used to encode the text; if it is Nil, the default encoding is used

RESULTS

ant handle to an annotation

ERRORS

#HPDF_INVALID_PAGE - An invalid page handle was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

#HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.20 page:CreateStampAnnot**NAME**

page:CreateStampAnnot – create stamp annotation object

SYNOPSIS

ant = page:CreateStampAnnot(**rect**, **stamp**, **text**, **encoder**)

FUNCTION

`page:CreateStampAnnot()` creates a new stamp annotation object for the page.

The `stamp` parameter must be one of the following constants:

```
#HPDF_STAMP_ANNOT_APPROVED
#HPDF_STAMP_ANNOT_EXPERIMENTAL
#HPDF_STAMP_ANNOT_NOTAPPROVED
#HPDF_STAMP_ANNOT_ASIS
#HPDF_STAMP_ANNOT_EXPIRED
#HPDF_STAMP_ANNOT_NOTFORPUBLICRELEASE
#HPDF_STAMP_ANNOT_CONFIDENTIAL
#HPDF_STAMP_ANNOT_FINAL
#HPDF_STAMP_ANNOT_SOLD
#HPDF_STAMP_ANNOT_DEPARTMENTAL
#HPDF_STAMP_ANNOT_FORCOMMENT
#HPDF_STAMP_ANNOT_TOPSECRET
#HPDF_STAMP_ANNOT_DRAFT
#HPDF_STAMP_ANNOT_FORPUBLICRELEASE
```

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

<code>rect</code>	a rectangle of the clickable area
<code>stamp</code>	stamp annotation type (see above for possible values)
<code>text</code>	the text to be displayed
<code>encoder</code>	an encoder handle which is used to encode the text; if it is <code>Nil</code> , the default encoding is used

RESULTS

<code>ant</code>	handle to an annotation
------------------	-------------------------

ERRORS

```
#HPDF_INVALID_PAGE - An invalid page handle was set.
#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
#HPDF_INVALID_ENCODER - An invalid encoder handle is specified.
```

15.21 `page:CreateStrikeOutAnnot`

NAME

`page:CreateStrikeOutAnnot` – create strike out annotation object

SYNOPSIS

```
ant = page:CreateStrikeOutAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateStrikeOutAnnot()` creates a new strike out annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

<code>rect</code>	a rectangle of the clickable area
<code>text</code>	the text to be displayed
<code>encoder</code>	an encoder handle which is used to encode the text; if it is <code>Nil</code> , the default encoding is used

RESULTS

<code>ant</code>	handle to an annotation
------------------	-------------------------

ERRORS

- `#HPDF_INVALID_PAGE` - An invalid page handle was set.
- `#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.
- `#HPDF_INVALID_ENCODER` - An invalid encoder handle is specified.

15.22 `page:CreateTextAnnot`

NAME

`page:CreateTextAnnot` – create text annotation object

SYNOPSIS

```
ant = page:CreateTextAnnot(rect, text, encoder)
```

FUNCTION

`page:CreateTextAnnot()` creates a new text annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

<code>rect</code>	a rectangle where the annotation is displayed
<code>text</code>	the text to be displayed
<code>encoder</code>	an encoder handle which is used to encode the text; if it is <code>Nil</code> , the default encoding is used

RESULTS

<code>ant</code>	handle to an annotation
------------------	-------------------------

ERRORS

- `#HPDF_INVALID_PAGE` - An invalid page handle was set.
- `#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.
- `#HPDF_INVALID_ENCODER` - An invalid encoder handle is specified.

15.23 page:CreateTextMarkupAnnot

NAME

page:CreateTextMarkupAnnot – create text markup annotation object

SYNOPSIS

```
ant = page:CreateTextMarkupAnnot(rect, text, encoder, subtype)
```

FUNCTION

`page:CreateTextMarkupAnnot()` creates a new text markup annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

The `subtype` parameter must be one of the following constants:

```
#HPDF_ANNOT_TEXT_NOTES
#HPDF_ANNOT_LINK
#HPDF_ANNOT_SOUND
#HPDF_ANNOT_FREE_TEXT
#HPDF_ANNOT_STAMP
#HPDF_ANNOT_SQUARE
#HPDF_ANNOT_CIRCLE
#HPDF_ANNOT_STRIKE_OUT
#HPDF_ANNOT_HIGHLIGHT
#HPDF_ANNOT_UNDERLINE
#HPDF_ANNOT_INK
#HPDF_ANNOT_FILE_ATTACHMENT
#HPDF_ANNOT_POPUP
#HPDF_ANNOT_3D
#HPDF_ANNOT_SQUIGGLY
#HPDF_ANNOT_LINE
#HPDF_ANNOT_PROJECTION
#HPDF_ANNOT_WIDGET
```

INPUTS

<code>rect</code>	a rectangle of the clickable area
<code>text</code>	the text to be displayed
<code>encoder</code>	an encoder handle which is used to encode the text; if it is <code>Nil</code> , the default encoding is used
<code>subtype</code>	subtype of annotation object (see above for possible values)

RESULTS

`ant` handle to an annotation

ERRORS

```
#HPDF_INVALID_PAGE - An invalid page handle was set.
#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
#HPDF_INVALID_ENCODER - An invalid encoder handle is specified.
```

15.24 page:CreateUnderlineAnnot

NAME

page:CreateUnderlineAnnot – create underline annotation object

SYNOPSIS

```
ant = page:CreateUnderlineAnnot(rect, text, encoder)
```

FUNCTION

page:CreateUnderlineAnnot() creates a new underline annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

<code>rect</code>	a rectangle of the clickable area
<code>text</code>	the text to be displayed
<code>encoder</code>	an encoder handle which is used to encode the text; if it is <code>Nil</code> , the default encoding is used

RESULTS

<code>ant</code>	handle to an annotation
------------------	-------------------------

ERRORS

- #HPDF_INVALID_PAGE - An invalid page handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
- #HPDF_INVALID_ENCODER - An invalid encoder handle is specified.

15.25 page:CreateURILinkAnnot

NAME

page:CreateURILinkAnnot – create web link annotation object

SYNOPSIS

```
ant = page:CreateURILinkAnnot(rect, uri)
```

FUNCTION

page:CreateURILinkAnnot() creates a new web link annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

<code>rect</code>	a rectangle of clickable area
<code>uri</code>	URL of destination to jump to

RESULTS

<code>ant</code>	handle to an annotation
------------------	-------------------------

ERRORS

- #HPDF_INVALID_PAGE - An invalid page handle was set.
- #HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

15.26 page:CreateWidgetAnnot

NAME

page:CreateWidgetAnnot – create widget annotation object

SYNOPSIS

```
ant = page:CreateWidgetAnnot(rect)
```

FUNCTION

page:CreateWidgetAnnot() creates a new widget annotation object for the page.

The `rect` parameter must be a table which contains `left`, `top`, `right`, and `bottom` fields that describe a rectangle.

INPUTS

`rect` a rectangle of the clickable area

RESULTS

`ant` handle to an annotation

15.27 page:CurveTo

NAME

page:CurveTo – append Bezier curve to path

SYNOPSIS

```
status = page:CurveTo(x1, y1, x2, y2, x3, y3)
```

FUNCTION

page:CurveTo() appends a Bezier curve to the current path using the control points (x1, y1) and (x2, y2) and (x3, y3), then sets the current point to (x3, y3).

INPUTS

`x1` x coordinate of control point #1

`y1` y coordinate of control point #1

`x2` x coordinate of control point #2

`y2` y coordinate of control point #2

`x3` x coordinate of curve destination point

`y3` y coordinate of curve destination point

RESULTS

`status` status code

15.28 page:CurveTo2

NAME

page:CurveTo2 – append Bezier curve to path

SYNOPSIS

```
status = page:CurveTo2(x2, y2, x3, y3)
```

FUNCTION

`page:CurveTo2()` appends a Bezier curve to the current path using the current point and (x2, y2) and (x3, y3) as control points. Then, the current point is set to (x3, y3).

INPUTS

x2	x coordinate of control point #1
y2	y coordinate of control point #1
x3	x coordinate of control point #2
y3	y coordinate of control point #2

RESULTS

status	status code
--------	-------------

15.29 page:CurveTo3

NAME

page:CurveTo3 – append Bezier curve to path

SYNOPSIS

```
status = page:CurveTo3(x1, y1, x3, y3)
```

FUNCTION

`page:CurveTo3()` appends a Bezier curve to the current path using two specified points. The point (x1, y1) and the point (x3, y3) are used as the control points for a Bezier curve and current point is moved to the point (x3, y3)

INPUTS

x1	x coordinate of control point #1
y1	y coordinate of control point #1
x3	x coordinate of control point #2
y3	y coordinate of control point #2

RESULTS

status	status code
--------	-------------

15.30 page:DrawImage

NAME

page:DrawImage – draw image to page

SYNOPSIS

```
status = page:DrawImage(image, x, y, width, height)
```

FUNCTION

page:DrawImage() shows an image in one operation.

INPUTS

image	the handle of an image object
x	horizontal coordinate for image
y	vertical coordinate for image
width	the width of the region where image is displayed
height	the height of the region where image is displayed

RESULTS

status	status code
--------	-------------

15.31 page:Ellipse

NAME

page:Ellipse – append ellipse to path

SYNOPSIS

```
status = page:Ellipse(x, y, xradius, yradius)
```

FUNCTION

page:Ellipse() appends an ellipse to the current path.

INPUTS

x	x center point of the ellipse
y	y center point of the ellipse
xradius	horizontal radius of the ellipse
yradius	vertical radius of the ellipse

RESULTS

status	status code
--------	-------------

15.32 page:EndPath

NAME

page:EndPath – end path

SYNOPSIS

```
status = page:EndPath()
```

FUNCTION

`page:EndPath()` ends the path object without filling or painting.

INPUTS

none

RESULTS

status status code

15.33 page:EndText

NAME

page:EndText – end a text object

SYNOPSIS

```
status = page:EndText()
```

FUNCTION

`page:EndText()` ends a text object.

INPUTS

none

RESULTS

status status code

15.34 page:EoClip

NAME

page:EoClip – modify clipping path using even-odd rule

SYNOPSIS

```
status = page:EoClip()
```

FUNCTION

`page:Clip()` modifies the current clipping path by intersecting it with the current path using the even-odd rule. The clipping path is only modified after the succeeding painting operator. To avoid painting the current path, use the function `page:EndPath()`.

Following painting operations will only affect the regions of the page contained by the clipping path. Initially, the clipping path includes the entire page. There is no way to enlarge the current clipping path, or to replace the clipping path with a new one. The functions `page:GSave()` and `page:GRestore()` may be used to save and restore the current graphics state, including the clipping path.

INPUTS

none

RESULTS

status status code

15.35 page:Eofill**NAME**

page:Eofill – fill current path using even-odd rule

SYNOPSIS

status = page:Eofill()

FUNCTION

page:Eofill() fills the current path using the even-odd rule.

INPUTS

none

RESULTS

status status code

15.36 page:EofillStroke**NAME**

page:EofillStroke – fill and paint current path using even-odd rule

SYNOPSIS

status = page:EofillStroke()

FUNCTION

page:EofillStroke() fills the current path using the even-odd rule, then paints the path.

INPUTS

none

RESULTS

status status code

15.37 page:ExecuteXObject**NAME**

page:ExecuteXObject – execute X object

SYNOPSIS

status = page:ExecuteXObject(xobj)

FUNCTION

page:ExecuteXObject() executes the specified X object.

INPUTS

xobj handle to an X object

RESULTS

status status code

15.38 page:Fill

NAME

page:Fill – fill current path

SYNOPSIS

status = page:Fill()

FUNCTION

`page:Fill()` fills the current path using the nonzero winding number rule.

INPUTS

none

RESULTS

status status code

15.39 page:FillStroke

NAME

page:FillStroke – fill and paint current path

SYNOPSIS

status = page:FillStroke()

FUNCTION

`page:FillStroke()` fills the current path using the nonzero winding number rule, then paints the path.

INPUTS

none

RESULTS

status status code

15.40 page:GetCharSpace

NAME

page:GetCharSpace – get current character spacing

SYNOPSIS

charspace = page:GetCharSpace()

FUNCTION

`page:GetCharSpace()` gets the current value of the page's character spacing.

INPUTS

none

RESULTS

`charspace`
current character spacing

15.41 page:GetCMYKFill**NAME**

`page:GetCMYKFill` – get CMYK filling color

SYNOPSIS

`t = page:GetCMYKFill()`

FUNCTION

`page:GetCMYKFill()` returns the current value of the page's filling color. `page:GetCMYKFill()` is valid only when the page's filling color space is `#HPDF_CS_DEVICE_CMYK`.

This function returns a table with the following fields initialized:

<code>C</code>	Cyan level of color.
<code>Y</code>	Yellow level of color.
<code>M</code>	Magenta level of color.
<code>K</code>	Black level of color.

All fields contain values between 0 and 1.

INPUTS

none

RESULTS

`t` current CMYK filling color

15.42 page:GetCMYKStroke**NAME**

`page:GetCMYKStroke` – get current CMYK stroking color

SYNOPSIS

`t = page:GetCMYKStroke()`

FUNCTION

`page:GetCMYKStroke()` returns the current value of the page's stroking color. `page:GetCMYKStroke()` is valid only when the page's stroking color space is `#HPDF_CS_DEVICE_CMYK`.

This function returns a table with the following fields initialized:

C	Cyan level of color.
Y	Yellow level of color.
M	Magenta level of color.
K	Black level of color.

All fields contain values between 0 and 1.

INPUTS

none

RESULTS

t current CMYK stroking color

15.43 page:GetCurrentFont

NAME

page:GetCurrentFont – get current font

SYNOPSIS

```
font = page:GetCurrentFont()
```

FUNCTION

page:GetCurrentFont() gets the handle of the page's current font.

INPUTS

none

RESULTS

font handle to a font

15.44 page:GetCurrentFontSize

NAME

page:GetCurrentFontSize – get current font size

SYNOPSIS

```
size = page:GetCurrentFontSize()
```

FUNCTION

page:GetCurrentFontSize() gets the size of the page's current font.

INPUTS

none

RESULTS

size current font size

15.45 page:GetCurrentPos

NAME

page:GetCurrentPos – get current path position

SYNOPSIS

```
x, y = page:GetCurrentPos()
```

FUNCTION

`page:GetCurrentPos()` gets the current position for path painting.

An application can invoke `page:GetCurrentPos()` only when graphics mode is `#HPDF_GMODE_PATH_OBJECT`.

INPUTS

none

RESULTS

x current x position

y current y position

15.46 page:GetCurrentTextPos

NAME

page:GetCurrentTextPos – get current text position

SYNOPSIS

```
x, y = page:GetCurrentTextPos()
```

FUNCTION

`page:GetCurrentTextPos()` gets the current position for drawing text.

An application can invoke `page:GetCurrentTextPos()` only when graphics mode is `#HPDF_GMODE_TEXT_OBJECT`.

INPUTS

none

RESULTS

x current x position

y current y position

15.47 page:GetDash

NAME

page:GetDash – get current dash pattern

SYNOPSIS

```
t = page:GetDash()
```

FUNCTION

`page:GetDash()` gets the current pattern of the page.

This method will return a table that has the following fields initialized:

`ptn` A table containing the individual on and off sections of the pattern.
`num_ptn` The number of elements in the `ptn` table.
`phase` The phase in which the pattern begins.

See [Section 15.81 \[page:SetDash\], page 150](#), for details.

INPUTS

none

RESULTS

`t` table containing the current dash pattern (see above)

15.48 page:GetFillingColorSpace**NAME**

`page:GetFillingColorSpace` – get filling color space

SYNOPSIS

`cs = page:GetFillingColorSpace()`

FUNCTION

`page:GetFillingColorSpace()` returns the current value of the page's filling color space. This will be one of `#HPDF_CS_DEVICE_GRAY`, `#HPDF_CS_DEVICE_RGB` or `#HPDF_CS_DEVICE_CMYK`.

INPUTS

none

RESULTS

`cs` current filling color space

15.49 page:GetFlat**NAME**

`page:GetFlat` – get current flatness

SYNOPSIS

`flat = page:GetFlat()`

FUNCTION

`page:GetFlat()` gets the current value of the page's flatness.

INPUTS

none

RESULTS

`flat` current flatness

15.50 page:GetGMode

NAME

page:GetGMode – get current graphics mode

SYNOPSIS

```
mode = page:GetGMode()
```

FUNCTION

page:GetGMode() gets the current graphics mode.

The following graphics modes are available:

```
#HPDF_GMODE_PAGE_DESCRIPTION
#HPDF_GMODE_PATH_OBJECT
#HPDF_GMODE_TEXT_OBJECT
#HPDF_GMODE_CLIPPING_PATH
#HPDF_GMODE_SHADING
#HPDF_GMODE_INLINE_IMAGE
#HPDF_GMODE_EXTERNAL_OBJECT
```

INPUTS

none

RESULTS

mode current graphics mode

15.51 page:GetGrayFill

NAME

page:GetGrayFill – get gray filling color

SYNOPSIS

```
gray = page:GetGrayFill()
```

FUNCTION

page:GetGrayFill() returns the current value of the page's filling color.

page:GetGrayFill() is valid only when the page's filling color space is #HPDF_CS_DEVICE_GRAY.

INPUTS

none

RESULTS

gray current gray filling color

15.52 page:GetGrayStroke

NAME

page:GetGrayStroke – get gray stroking color

SYNOPSIS

```
gray = page:GetGrayStroke()
```

FUNCTION

`page:GetGrayStroke()` returns the current value of the page's stroking color. `page:GetGrayStroke()` is valid only when the page's stroking color space is `#HPDF_CS_DEVICE_GRAY`.

INPUTS

none

RESULTS

`gray` current gray stroking color

15.53 `page:GetGStateDepth`

NAME

`page:GetGStateDepth` – get graphics state stack

SYNOPSIS

```
d = page:GetGStateDepth()
```

FUNCTION

`page:GetGStateDepth()` returns the number of the page's graphics state stack.

INPUTS

none

RESULTS

`d` current graphics state stack

15.54 `page:GetHeight`

NAME

`page:GetHeight` – get page height

SYNOPSIS

```
h = page:GetHeight()
```

FUNCTION

`page:GetHeight()` gets the height of a page.

INPUTS

none

RESULTS

`h` page height

15.55 page:GetHorizontalScaling

NAME

page:GetHorizontalScaling – get current horizontal scaling

SYNOPSIS

```
s = page:GetHorizontalScaling()
```

FUNCTION

`page:GetHorizontalScaling()` returns the current value of the page's horizontal scaling for drawing text.

INPUTS

none

RESULTS

s horizontal scaling value

15.56 page:GetLineCap

NAME

page:GetLineCap – get current line cap style

SYNOPSIS

```
cap = page:GetLineCap()
```

FUNCTION

`page:GetLineCap()` gets the current line cap style of the page.

See [Section 15.89 \[page:SetLineCap\]](#), [page 153](#), for a list of available line cap styles.

INPUTS

none

RESULTS

cap current line cap style

15.57 page:GetLineJoin

NAME

page:GetLineJoin – get current line join style

SYNOPSIS

```
linejoin = page:GetLineJoin()
```

FUNCTION

`page:GetLineJoin()` gets the current line join style of the page.

See [Section 15.90 \[page:SetLineJoin\]](#), [page 153](#), for a list of available line join styles.

INPUTS

none

RESULTS

linejoin current line join style

15.58 page:GetLineWidth**NAME**

page:GetLineWidth – get line width of page

SYNOPSIS

```
w = page:GetLineWidth()
```

FUNCTION

`page:GetLineWidth()` gets the current line width of the page.

INPUTS

none

RESULTS

w current line width

15.59 page:GetMiterLimit**NAME**

page:GetMiterLimit – get current miter limit

SYNOPSIS

```
limit = page:GetMiterLimit()
```

FUNCTION

`page:GetMiterLimit()` gets the current value of the page's miter limit.

INPUTS

none

RESULTS

limit current miter limit

15.60 page:GetRGBFill**NAME**

page:GetRGBFill – get current RGB filling color

SYNOPSIS

```
t = page:GetRGBFill()
```

FUNCTION

`page:GetRGBFill()` returns the current value of the page's filling color. `page:GetRGBFill()` is valid only when the page's filling color space is `#HPDF_CS_DEVICE_RGB`.

This function returns a table with the following fields initialized:

R Red level of color.
 G Green level of color.
 B Blue level of color.

All fields contain values between 0 and 1.

INPUTS

none

RESULTS

t current RGB filling color

15.61 page:GetRGBStroke

NAME

page:GetRGBStroke – get RGB stroking color

SYNOPSIS

t = page:GetRGBStroke()

FUNCTION

page:GetRGBStroke() returns the current value of the page's stroking color. page:GetRGBStroke() is valid only when the page's stroking color space is #HPDF_CS_DEVICE_RGB.

This function returns a table with the following fields initialized:

R Red level of color.
 G Green level of color.
 B Blue level of color.

All fields contain values between 0 and 1.

INPUTS

none

RESULTS

t current RGB stroking color

15.62 page:GetStrokingColorSpace

NAME

page:GetStrokingColorSpace – get stroking color space

SYNOPSIS

cs = page:GetStrokingColorSpace()

FUNCTION

`page:GetStrokingColorSpace()` returns the current value of the page's stroking color space. This will be one of `#HPDF_CS_DEVICE_GRAY`, `#HPDF_CS_DEVICE_RGB` or `#HPDF_CS_DEVICE_CMYK`.

INPUTS

none

RESULTS

`cs` current stroking color space

15.63 `page:GetTextLeading`

NAME

`page:GetTextLeading` – get current line spacing

SYNOPSIS

`l = page:GetTextLeading()`

FUNCTION

`page:GetTextLeading()` returns the current value of the page's line spacing.

INPUTS

none

RESULTS

`l` current line spacing

15.64 `page:GetTextMatrix`

NAME

`page:GetTextMatrix` – get current text transformation matrix

SYNOPSIS

`m = page:GetTextMatrix()`

FUNCTION

`page:GetTextMatrix()` gets the current text transformation matrix of the page.

This method will return the transformation matrix in a table with the following fields initialized:

<code>a</code>	Scaling x coordinate
<code>b</code>	Rotation x coordinate
<code>c</code>	Rotation y coordinate
<code>d</code>	Scaling y coordinate
<code>x</code>	Translation x coordinate
<code>y</code>	Translation y coordinate

INPUTS

none

RESULTS

m current text transformation matrix

15.65 page:GetTextRenderingMode**NAME**

page:GetTextRenderingMode – get current text rendering mode

SYNOPSIS

mode = page:GetTextRenderingMode()

FUNCTION

page:GetTextRenderingMode() returns the current value of the page's text rendering mode.

See [Section 15.100 \[page:SetTextRenderingMode\]](#), page 159, for a list of available text rendering modes.

INPUTS

none

RESULTS

mode current text rendering mode

15.66 page:GetTextRise**NAME**

page:GetTextRise – get current text rising

SYNOPSIS

rise = page:GetTextRise()

FUNCTION

page:GetTextRise() returns the current value of the page's text rising.

INPUTS

none

RESULTS

rise current text rising

15.67 page:GetTransMatrix**NAME**

page:GetTransMatrix – get current transformation matrix

SYNOPSIS

```
m = page:GetTransMatrix()
```

FUNCTION

`page:GetTransMatrix()` gets the current transformation matrix of the page.

This method will return the transformation matrix in a table with the following fields initialized:

a	Scaling x coordinate
b	Rotation x coordinate
c	Rotation y coordinate
d	Scaling y coordinate
x	Translation x coordinate
y	Translation y coordinate

INPUTS

none

RESULTS

m	transformation matrix
---	-----------------------

15.68 page:GetWidth**NAME**

`page:GetWidth` – get page width

SYNOPSIS

```
w = page:GetWidth()
```

FUNCTION

`page:GetWidth()` gets the width of the page.

INPUTS

none

RESULTS

w	page width
---	------------

15.69 page:GetWordSpace**NAME**

`page:GetWordSpace` – get current word spacing

SYNOPSIS

```
wordspace = page:GetWordSpace()
```

FUNCTION

`page:GetWordSpace()` returns the current value of the page's word spacing.

INPUTS

none

RESULTS

`wordspace`
current word spacing

15.70 `page:GRestore`

NAME`page:GRestore` – restore graphics state**SYNOPSIS**`status = page:GRestore()`**FUNCTION**`page:GRestore()` restore the graphics state which is saved by `page:GSave()`.**INPUTS**

none

RESULTS`status` status code

15.71 `page:GSave`

NAME`page:GSave` – save current graphics parameters**SYNOPSIS**`status = page:GSave()`**FUNCTION**

`page:GSave()` saves the page's current graphics parameters. An application can invoke `page:GSave()` up to 28 times and can restore the saved parameter by invoking `page:GRestore()`.

The parameters that are saved by `page:GSave()` are:

- Character Spacing
- Clipping Path
- Dash Mode
- Filling Color
- Flatness
- Font
- Font Size
- Horizontal Scaling
- Line Width

- Line Cap Style
- Line Join Style
- Miter Limit
- Rendering Mode
- Stroking Color
- Text Leading
- Text Rise
- Transformation Matrix
- Word Spacing

INPUTS

none

RESULTS

`status` status code

15.72 page:LineTo**NAME**

`page:LineTo` – append line to path

SYNOPSIS

`status = page:LineTo(x, y)`

FUNCTION

`page:LineTo()` appends a line from the current point to the specified point.

INPUTS

`x` x coordinate of end point of the path

`y` y coordinate of end point of the path

RESULTS

`status` status code

15.73 page:MeasureText**NAME**

`page:MeasureText` – get byte length of text

SYNOPSIS

`bl, rw = page:MeasureText(text, width, wordwrap)`

FUNCTION

`page:MeasureText()` calculates the byte length which can be included within the specified width.

The `wordwrap` parameter configures how words should be wrapped: Suppose there are three words: "ABCDE", "FGH", and "IJKL". Also, suppose the substring until "J"

can be included within the width (12 bytes). If `wordwrap` is `False` the function returns 12. If `wordwrap` parameter is `True`, it returns 10 (the end of the previous word).

INPUTS

`text` the text whose length to compute
`width` the width of the area to put the text
`wordwrap` boolean that says whether wordwrapping should be used

RESULTS

`bl` byte length of text
`rw` real width of text

15.74 page:MoveTextPos**NAME**

`page:MoveTextPos` – change current text position

SYNOPSIS

`status = page:MoveTextPos(x, y[, lead])`

FUNCTION

`page:MoveTextPos()` changes the current text position, using the specified offset values. If the current text position is (x1, y1), the new text position will be (x1 + x, y1 + y). If the optional argument `lead` is set to `True`, the text leading is set to -y.

INPUTS

`x` x offset for text
`y` y offset for text
`lead` optional: whether or not to set text leading to -y

RESULTS

`status` status code

15.75 page:MoveTo**NAME**

`page:MoveTo` – start new subpath

SYNOPSIS

`status = page:MoveTo(x, y)`

FUNCTION

`page:MoveTo()` starts a new subpath and move the current point for drawing path.
`page:MoveTo()` sets the start point for the path to the point (x, y).

INPUTS

`x` x start point for drawing path

`y` `y` start point for drawing path

RESULTS

`status` status code

15.76 `page:MoveToNextLine`

NAME

`page:MoveToNextLine` – move current position to next line

SYNOPSIS

`status` = `page:MoveToNextLine()`

FUNCTION

`page:MoveToNextLine()` moves current position for the drawing text depending on current text showing point and text leading. The new position is calculated with current text transition matrix.

Returns `#HPDF_OK` on success. Otherwise, returns an error code and the error handler is invoked.

INPUTS

none

RESULTS

`status` status code

15.77 `page:Rectangle`

NAME

`page:Rectangle` – append rectangle to path

SYNOPSIS

`status` = `page:Rectangle(x, y, width, height)`

FUNCTION

`page:Rectangle()` appends a rectangle to the current path.

INPUTS

`x` `x` coordinate of lower left point of the rectangle

`y` `y` coordinate of lower left point of the rectangle

`width` width of the rectangle

`height` height of the rectangle

RESULTS

`status` status code

15.78 page:SetCharSpace

NAME

page:SetCharSpace – set character spacing

SYNOPSIS

```
status = page:SetCharSpace(value)
```

FUNCTION

`page:SetCharSpace()` sets the character spacing for text.

INPUTS

value the character spacing (initial value is 0)

RESULTS

status status code

15.79 page:SetCMYKFill

NAME

page:SetCMYKFill – set CMYK filling color

SYNOPSIS

```
status = page:SetCMYKFill(c, m, y, k)
```

FUNCTION

`page:SetCMYKFill()` sets the filling color. The individual parameters must all be between 0 and 1.

INPUTS

c level of cyan
m level of magenta
y level of yellow
k level of black

RESULTS

status status code

15.80 page:SetCMYKStroke

NAME

page:SetCMYKStroke – set CMYK stroking color

SYNOPSIS

```
status = page:SetCMYKStroke(c, m, y, k)
```

FUNCTION

`page:SetCMYKStroke()` sets the stroking color. The individual parameters must all be between 0 and 1.

INPUTS

c	level of cyan
m	level of magenta
y	level of yellow
k	level of black

RESULTS

status	status code
--------	-------------

15.81 page:SetDash**NAME**

page:SetDash – set dash pattern for lines

SYNOPSIS

```
status = page:SetDash([pattern, phase])
```

FUNCTION

`page:SetDash()` sets the dash pattern for lines in the page. `pattern` needs to be a table containing between 0 and 8 elements of dashes and gaps. When called without parameters, line dashing will be disabled.

Here are some common patterns:

```
page:SetDash({3}, 1)
page:SetDash({7,3}, 2)
page:SetDash({8,7,2,7}, 0)
```

INPUTS

pattern	optional: pattern of dashes and gaps used to stroke paths
phase	optional: the phase in which the pattern begins (default is 0)

RESULTS

status	status code
--------	-------------

15.82 page:SetExtGState**NAME**

page:SetExtGState – apply extended graphics state

SYNOPSIS

```
status = page:SetExtGState(extgstate)
```

FUNCTION

`page:SetExtGState()` applies the graphics state to the page.

INPUTS

extgstate	the handle of an extended graphics state object
-----------	-------------------------------------------------

RESULTS

`status` status code

15.83 page:SetFlat**NAME**

`page:SetFlat` – set current flatness

SYNOPSIS

`status = page:SetFlat(flatness)`

FUNCTION

`page:SetFlat()` sets the current value of the page's flatness.

INPUTS

`flatness` desired flatness

RESULTS

`status` status code

15.84 page:SetFontAndSize**NAME**

`page:SetFontAndSize` – set font and size

SYNOPSIS

`status = page:SetFontAndSize(font, size)`

FUNCTION

`page:SetFontAndSize()` sets the type of font and size leading.

INPUTS

`font` the handle of a font object

`size` the size of a font

RESULTS

`status` status code

15.85 page:SetGrayFill**NAME**

`page:SetGrayFill` – set gray filling color

SYNOPSIS

`status = page:SetGrayFill(gray)`

FUNCTION

`page:SetGrayFill()` sets the filling color.

INPUTS

value the value of the gray level between 0 and 1

RESULTS

status status code

15.86 page:SetGrayStroke

NAME

page:SetGrayStroke – set gray stroking color

SYNOPSIS

```
status = page:SetGrayStroke(gray)
```

FUNCTION

`page:SetGrayStroke()` sets the stroking color.

INPUTS

value the value of the gray level between 0 and 1

RESULTS

status status code

15.87 page:SetHeight

NAME

page:SetHeight – set page height

SYNOPSIS

```
status = page:SetHeight(value)
```

FUNCTION

`page:SetHeight()` changes the height of a page.

INPUTS

value the new page height; valid values are between 3 and 14400

RESULTS

status status code

ERRORS

#HPDF_INVALID_PAGE - An invalid page handle was set.

#HPDF_PAGE_INVALID_SIZE - An invalid size was set.

#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.

15.88 page:SetHorizontalScaling

NAME

page:SetHorizontalScaling – set horizontal scaling for text

SYNOPSIS

```
status = page:SetHorizontalScaling(value)
```

FUNCTION

page:SetHorizontalScaling() sets the horizontal scaling for text.

INPUTS

value the value of horizontal scaling (initially 100)

RESULTS

status status code

15.89 page:SetLineCap

NAME

page:SetLineCap – set line cap style

SYNOPSIS

```
status = page:SetLineCap(linecap)
```

FUNCTION

page:SetLineCap() sets the shape to be used at the ends of lines.

The linecap parameter must be one of the following constants:

#HPDF_BUTT_END

Line is squared off at path endpoint

#HPDF_ROUND_END

End of line becomes a semicircle whose center is at path endpoint

#HPDF_PROJECTING_SQUARE_END

Line continues beyond endpoint, goes on half the endpoint stroke width

INPUTS

linecap the desired line cap style (see above)

RESULTS

status status code

15.90 page:SetLineJoin

NAME

page:SetLineJoin – set line join style

SYNOPSIS

```
status = page:SetLineJoin(linejoin)
```

FUNCTION

`page:SetLineJoin()` Sets the line join style in the page.

The `linejoin` parameter must be one of the following constants:

`#HPDF_MITER_JOIN`

Use miter join (a sharp angled corner). This is the default join mode.

`#HPDF_ROUND_JOIN`

Join lines by drawing their ends as circles. This gives a thick pen impression.

`#HPDF_BEVEL_JOIN`

Join lines by cutting off the line ends at the half of the line width.

INPUTS

`linejoin` the desired line join style (see above)

RESULTS

`status` status code

15.91 page:SetLineWidth**NAME**

`page:SetLineWidth` – set stroking width

SYNOPSIS

`status = page:SetLineWidth(linewidth)`

FUNCTION

`page:SetLineWidth()` sets the width of the line used to stroke a path.

INPUTS

`linewidth`

the line width to use (default is 1)

RESULTS

`status` status code

15.92 page:SetMiterLimit**NAME**

`page:SetMiterLimit` – set miter limit

SYNOPSIS

`status = page:SetMiterLimit(miterlimit)`

FUNCTION

Sets the miter limit. This defaults to 10.

INPUTS

`miterlimit`

desired miter limit

RESULTS

`status` status code

15.93 page:SetRGBFill**NAME**

`page:SetRGBFill` – set RGB fill color

SYNOPSIS

```
status = page:SetRGBFill(r, g, b)
```

FUNCTION

`page:SetRGBFill()` sets the filling color. The individual color components must be between 0 and 1.

INPUTS

`r` red level of new color
`g` green level of new color
`b` blue level of new color

RESULTS

`status` status code

15.94 page:SetRGBStroke**NAME**

`page:SetRGBStroke` – set RGB stroking color

SYNOPSIS

```
status = page:SetRGBStroke(r, g, b)
```

FUNCTION

`page:SetRGBStroke()` sets the stroking color. The individual color components must be between 0 and 1.

INPUTS

`r` red level of new color
`g` green level of new color
`b` blue level of new color

RESULTS

`status` status code

15.95 page:SetRotate

NAME

page:SetRotate – set page rotation

SYNOPSIS

```
status = page:SetRotate(angle)
```

FUNCTION

`page:SetRotate()` sets the rotation angle of the page.

INPUTS

`angle` the rotation angle of the page; it must be a multiple of 90 degrees

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_PAGE` - An invalid page handle was set.

`#HPDF_PAGE_INVALID_ROTATE_VALUE` - An invalid rotation angle was set.

15.96 page:SetSize

NAME

page:SetSize – set page size and direction

SYNOPSIS

```
status = page:SetSize(size, direction)
```

FUNCTION

`page:SetSize()` changes the size and direction of a page to a predefined size.

The `size` parameter must be one of the following constants:

`#HPDF_PAGE_SIZE_LETTER`

8.5 x 11 inches (612 x 792 pixels)

`#HPDF_PAGE_SIZE_LEGAL`

8.5 x 14 inches (612 x 1008 pixels)

`#HPDF_PAGE_SIZE_A3`

297 x 420 mm (841.89 x 1199.551 pixels)

`#HPDF_PAGE_SIZE_A4`

210 x 297 mm (595.276 x 841.89 pixels)

`#HPDF_PAGE_SIZE_A5`

148 x 210 mm (419.528 x 595.276 pixels)

`#HPDF_PAGE_SIZE_B4`

250 x 353 mm (708.661 x 1000.63 pixels)

`#HPDF_PAGE_SIZE_B5`

176 x 250 mm (498.898 x 708.661 pixels)

```
#HPDF_PAGE_SIZE_EXECUTIVE
    7.25 x 10.5 inches (522 x 756 pixels)

#HPDF_PAGE_SIZE_US4x6
    4 x 6 inches (288 x 432 pixels)

#HPDF_PAGE_SIZE_US4x8
    4 x 8 inches (288 x 576 pixels)

#HPDF_PAGE_SIZE_US5x7
    5 x 7 inches (360 x 504 pixels)

#HPDF_PAGE_SIZE_COMM10
    4.125 x 9.5 inches (297 x 684 pixels)
```

The `direction` parameter must be one of the following constants:

```
#HPDF_PAGE_PORTRAIT
    Set the longer value to vertical.

#HPDF_PAGE_LANDSCAPE
    Set the longer value to horizontal.
```

INPUTS

```
size      predefined page size value (see above)

direction the direction of the page (see above for possible values)
```

RESULTS

```
status    status code
```

ERRORS

```
#HPDF_INVALID_PAGE - An invalid page handle was set.
#HPDF_PAGE_INVALID_SIZE - An invalid size was set.
#HPDF_PAGE_INVALID_DIRECTION - An invalid direction was set.
#HPDF_FAILED_TO_ALLOC_MEM - Memory allocation failed.
```

15.97 page:SetSlideShow

NAME

`page:SetSlideShow` – set page transition mode

SYNOPSIS

```
status = page:SetSlideShow(type, disptime, transtime)
```

FUNCTION

`page:SetSlideShow()` configures the setting for slide transition of the page. The `disptime` specifies the display duration of the page in seconds whereas the `transtime` parameter must be set to the duration of the transition effect in seconds.

The `type` parameter configures the actual effect and can be one of the following values:

```
#HPDF_TS_WIPE_RIGHT
```

```

#HPDF_TS_WIPE_UP
#HPDF_TS_WIPE_LEFT
#HPDF_TS_WIPE_DOWN
#HPDF_TS_BARN_DOORS_HORIZONTAL_OUT
#HPDF_TS_BARN_DOORS_HORIZONTAL_IN
#HPDF_TS_BARN_DOORS_VERTICAL_OUT
#HPDF_TS_BARN_DOORS_VERTICAL_IN
#HPDF_TS_BOX_OUT
#HPDF_TS_BOX_IN
#HPDF_TS_BLINDS_HORIZONTAL
#HPDF_TS_BLINDS_VERTICAL
#HPDF_TS DISSOLVE
#HPDF_TS_GLITTER_RIGHT
#HPDF_TS_GLITTER_DOWN
#HPDF_TS_GLITTER_TOP_LEFT_TO_BOTTOM_RIGHT
#HPDF_TS_REPLACE

```

INPUTS

`type` the transition style (see above for possible values)

`disptime` the display duration of the page (in seconds)

`transtime`
the duration of the transition effect (in seconds)

RESULTS

`status` status code

15.98 page:SetTextLeading**NAME**

`page:SetTextLeading` – set text leading

SYNOPSIS

```
status = page:SetTextLeading(value)
```

FUNCTION

`page:SetTextLeading()` sets the text leading (line spacing) for showing text.

INPUTS

`value` the value of text leading (initial value is 0)

RESULTS

`status` status code

15.99 page:SetTextMatrix

NAME

page:SetTextMatrix – set text transformation matrix

SYNOPSIS

```
status = page:SetTextMatrix(a, b, c, d, x, y)
```

FUNCTION

`page:SetTextMatrix()` sets a transformation matrix for text to be drawn in using `page:ShowText()`. The function `page:TextRect()` does not use the active text matrix.

Returns `#HPDF_OK` on success, otherwise an error code.

INPUTS

a	scaling x coordinate
b	rotation x coordinate
c	rotation y coordinate
d	scaling y coordinate
x	translation x coordinate
y	translation y coordinate

RESULTS

status	status code
--------	-------------

15.100 page:SetTextRenderingMode

NAME

page:SetTextRenderingMode – set text rendering mode

SYNOPSIS

```
status = page:SetTextRenderingMode(mode)
```

FUNCTION

`page:SetTextRenderingMode()` sets the text rendering mode.

The mode parameter must be one of the following constants:

```
#HPDF_FILL  
#HPDF_STROKE  
#HPDF_FILL_THEN_STROKE  
#HPDF_INVISIBLE  
#HPDF_FILL_CLIPPING  
#HPDF_STROKE_CLIPPING  
#HPDF_FILL_STROKE_CLIPPING  
#HPDF_CLIPPING
```

The default text rendering mode is `#HPDF_FILL`.

INPUTS

mode	the text rendering mode (see above for possible modes)
------	--------------------------------------------------------

RESULTS

`status` status code

15.101 page:SetTextRise**NAME**

`page:SetTextRise` – modulate y position of text

SYNOPSIS

```
status = page:SetTextRise(value)
```

FUNCTION

`page:SetTextRise()` moves the text position in vertical direction by the amount of `value`. Useful for making subscripts or superscripts.

INPUTS

`value` text rise, in user space units

RESULTS

`status` status code

15.102 page:SetWidth**NAME**

`page:SetWidth` – set page width

SYNOPSIS

```
status = page:SetWidth(value)
```

FUNCTION

`page:SetWidth()` changes the width of a page.

INPUTS

`value` the new page width; valid values are between 3 and 14400

RESULTS

`status` status code

ERRORS

`#HPDF_INVALID_PAGE` - An invalid page handle was set.

`#HPDF_PAGE_INVALID_SIZE` - An invalid size was set.

`#HPDF_FAILED_TO_ALLOC_MEM` - Memory allocation failed.

15.103 page:SetWordSpace

NAME

page:SetWordSpace – set word spacing

SYNOPSIS

```
status = page:SetWordSpace(value)
```

FUNCTION

`page:SetWordSpace()` sets the word spacing for text.

INPUTS

value the value of word spacing (initial value is 0)

RESULTS

status status code

15.104 page:SetZoom

NAME

page:SetZoom – set page zoom

SYNOPSIS

```
status = page:SetZoom(zoom)
```

FUNCTION

`page:SetZoom()` sets the zoom factor for the page.

INPUTS

zoom the desired zoom setting

RESULTS

status status code

15.105 page:ShowText

NAME

page:ShowText – print text

SYNOPSIS

```
status = page:ShowText(text)
```

FUNCTION

`page:ShowText()` prints the text at the current position on the page.

INPUTS

text the text to print

RESULTS

status status code

15.106 page:ShowTextNextLine

NAME

page:ShowTextNextLine – break line and print text

SYNOPSIS

```
status = page:ShowTextNextLine(text[, wordspace, charspace])
```

FUNCTION

`page:ShowTextNextLine()` moves the current text position to the start of the next line, then prints the text at the current position on the page. If the optional arguments `wordspace` and `charspace` are specified, this method will also set the word and character spacing before printing the text.

INPUTS

`text` the text to print

`wordspace`
 optional: word spacing for text

`charspace`
 optional: char spacing for text

RESULTS

`status` status code

15.107 page:Stroke

NAME

page:Stroke – stroke current path

SYNOPSIS

```
status = page:Stroke()
```

FUNCTION

`page:Stroke()` paints the current path.

INPUTS

none

RESULTS

`status` status code

15.108 page:TextOut

NAME

page:TextOut – print text at position

SYNOPSIS

```
status = page:TextOut(xpos, ypos, text)
```

FUNCTION

`page:TextOut()` prints the text on the specified position.

INPUTS

`xpos` x position where the text is to be displayed
`ypos` y position where the text is to be displayed
`text` the text to show

RESULTS

`status` status code

15.109 page:TextRect**NAME**

`page:TextRect` – print text inside region

SYNOPSIS

`status, len = page:TextRect(left, top, right, bottom, text, align)`

FUNCTION

`page:TextRect()` prints the text inside the specified region.

The `align` parameter must be one of the following constants:

`#HPDF_TALIGN_LEFT`

The text is aligned to left.

`#HPDF_TALIGN_RIGHT`

The text is aligned to right.

`#HPDF_TALIGN_CENTER`

The text is centered.

`#HPDF_TALIGN_JUSTIFY`

Add spaces between the words to justify both left and right side.

INPUTS

`left` left coordinate of region
`top` top coordinate of region
`right` right coordinate of region
`bottom` bottom coordinate of region
`text` the text to show
`align` the alignment of the text (one of the following)

RESULTS

`status` status code
`len` the number of characters printed in the area

15.110 `page:TextWidth`

NAME

`page:TextWidth` – get text width

SYNOPSIS

```
w = page:TextWidth(text)
```

FUNCTION

`page:TextWidth()` gets the width of the text in the current font size, character spacing and word spacing.

INPUTS

`text` the text whose width to get

RESULTS

`w` text width

Appendix A Licenses

A.1 LibHaru license

Copyright (C) 1999-2006 Takeshi Kanno

Copyright (C) 2007-2009 Antony Dovgal

This software is provided 'as-is', without any express or implied warranty.

In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

A.2 LuaHPDF license

LuaHPDF is Copyright 2007-2013 by Kurt Jung.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.3 PDFium license

Copyright 2014 PDFium Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index

A

annot:SetBorderStyle	43
annot:SetCMYKColor	43
annot:SetFreeTextAnnot2PointCalloutLine	44
annot:SetFreeTextAnnot3PointCalloutLine	45
annot:SetFreeTextAnnotDefaultStyle	45
annot:SetFreeTextAnnotLineEndingStyle	45
annot:SetGrayColor	46
annot:SetLineAnnotCaption	47
annot:SetLineAnnotLeader	47
annot:SetLineAnnotPosition	48
annot:SetLinkAnnotBorderStyle	48
annot:SetLinkAnnotHighlightMode	49
annot:SetMarkupAnnotCloudEffect	50
annot:SetMarkupAnnotCreationDate	50
annot:SetMarkupAnnotIntent	51
annot:SetMarkupAnnotInteriorCMYKColor	51
annot:SetMarkupAnnotInteriorGrayColor	52
annot:SetMarkupAnnotInteriorRGBColor	52
annot:SetMarkupAnnotInteriorTransparent	53
annot:SetMarkupAnnotPopup	53
annot:SetMarkupAnnotQuadPoints	54
annot:SetMarkupAnnotRectDiff	54
annot:SetMarkupAnnotSubject	55
annot:SetMarkupAnnotTitle	55
annot:SetMarkupAnnotTransparency	56
annot:SetNoColor	56
annot:SetPopupAnnotOpened	56
annot:SetRGBColor	57
annot:SetTextAnnotIcon	57
annot:SetTextAnnotOpened	58

D

dest:SetFit	61
dest:SetFitB	61
dest:SetFitBH	61
dest:SetFitBV	62
dest:SetFitH	62
dest:SetFitR	63
dest:SetFitV	63
dest:SetXYZ	64
doc:AddPage	65
doc:AddPageLabel	65
doc:AttachFile	66
doc>CreateExtGState	66
doc>CreateImageFromBrush	67
doc>CreateImageFromMem	68
doc>CreateOutline	69
doc:Free	69
doc:GetCurrentEncoder	69
doc:GetCurrentPage	70
doc:GetEncoder	70
doc:GetError	71

doc:GetErrorDetail	71
doc:GetFont	72
doc:GetInfoAttr	72
doc:GetPageByIndex	73
doc:GetPageLayout	73
doc:GetPageMode	74
doc:GetViewerPreference	74
doc:InsertPage	75
doc:LoadFont	75
doc:LoadJPEGImage	76
doc:LoadPNGImage	77
doc:LoadRawImage	78
doc:LoadTTFont	79
doc:LoadType1Font	80
doc:ResetError	80
doc:SaveToFile	81
doc:SetCompressionMode	81
doc:SetCurrentEncoder	82
doc:SetEncryptionMode	82
doc:SetInfoAttr	83
doc:SetInfoDateAttr	84
doc:SetOpenAction	85
doc:SetPageLayout	85
doc:SetPageMode	86
doc:SetPagesConfiguration	87
doc:SetPassword	88
doc:SetPermission	88
doc:SetViewerPreference	89
doc:UseCNSEncodings	90
doc:UseCNSFonts	90
doc:UseCNTEncodings	91
doc:UseCNTFonts	92
doc:UseJPEncodings	92
doc:UseJPFonts	93
doc:UseKREncodings	94
doc:UseKRFonts	95
doc:UseUTFEncodings	96

E

encoder:GetByteType	97
encoder:GetType	97
encoder:GetUnicode	97
encoder:GetWritingMode	98
extgs:SetAlphaFill	99
extgs:SetAlphaStroke	99
extgs:SetBlendMode	100

F

font:GetAscent	101
font:GetBBox	101
font:GetCapHeight	101
font:GetDescent	102
font:GetEncodingName	102
font:GetFontName	102
font:GetUnicodeWidth	103
font:GetXHeight	103
font:MeasureText	103
font:TextWidth	104

I

image:AddSMask	107
image:GetBitsPerComponent	107
image:GetColorSpace()	107
image:GetHeight	108
image:GetSize	108
image:GetWidth	108
image:SetColorMask	109
image:SetMaskImage	109

O

outline:SetDestination	111
outline:SetOpened	111

P

page:Arc	113
page:BeginText	113
page:Circle	113
page:Clip	114
page:ClosePath	114
page:ClosePathEofillStroke	115
page:ClosePathFillStroke	115
page:ClosePathStroke	116
page:Concat	116
page>CreateCircleAnnot	117
page>CreateDestination	118
page>CreateFreeTextAnnot	118
page>CreateHighlightAnnot	119
page>CreateLineAnnot	119
page>CreateLinkAnnot	120
page>CreatePopupAnnot	120
page>CreateProjectionAnnot	121
page>CreateSquareAnnot	121
page>CreateSquigglyAnnot	122
page>CreateStampAnnot	122
page>CreateStrikeOutAnnot	123
page>CreateTextAnnot	124
page>CreateTextMarkupAnnot	124
page>CreateUnderlineAnnot	125
page>CreateURILinkAnnot	126
page>CreateWidgetAnnot	126
page:CurveTo	127
page:CurveTo2	127

page:CurveTo3	128
page:DrawImage	128
page:Ellipse	129
page:EndPath	129
page:EndText	130
page:EoClip	130
page:Eofill	131
page:EofillStroke	131
page:ExecuteXObject	131
page:Fill	132
page:FillStroke	132
page:GetCharSpace	132
page:GetCMYKFill	133
page:GetCMYKStroke	133
page:GetCurrentFont	134
page:GetCurrentFontSize	134
page:GetCurrentPos	134
page:GetCurrentTextPos	135
page:GetDash	135
page:GetFillingColorSpace	136
page:GetFlat	136
page:GetGMode	136
page:GetGrayFill	137
page:GetGrayStroke	137
page:GetGStateDepth	138
page:GetHeight	138
page:GetHorizontalScaling	138
page:GetLineCap	139
page:GetLineJoin	139
page:GetLineWidth	140
page:GetMiterLimit	140
page:GetRGBFill	140
page:GetRGBStroke	141
page:GetStrokingColorSpace	141
page:GetTextLeading	142
page:GetTextMatrix	142
page:GetTextRenderingMode	143
page:GetTextRise	143
page:GetTransMatrix	143
page:GetWidth	144
page:GetWordSpace	144
page:GRestore	145
page:GSave	145
page:LineTo	146
page:MeasureText	146
page:MoveTextPos	147
page:MoveTo	147
page:MoveToNextLine	148
page:Rectangle	148
page:SetCharSpace	148
page:SetCMYKFill	149
page:SetCMYKStroke	149
page:SetDash	150
page:SetExtGState	150
page:SetFlat	151
page:SetFontAndSize	151
page:SetGrayFill	151
page:SetGrayStroke	152

page:SetHeight	152	pdf.FindNext	22
page:SetHorizontalScaling	152	pdf.FindPrev	23
page:SetLineCap	153	pdf.FindStart	24
page:SetLineJoin	153	pdf.FreePage	24
page:SetLineWidth	154	pdf.GetBookmarks	25
page:SetMiterLimit	154	pdf.GetBoundedText	26
page:SetRGBFill	155	pdf.GetBrush	26
page:SetRGBStroke	155	pdf.GetBrushFromPage	28
page:SetRotate	155	pdf.GetCharBox	29
page:SetSize	156	pdf.GetCharIndexAtPos	29
page:SetSlideShow	157	pdf.GetCharOrigin	30
page:SetTextLeading	158	pdf.GetCropBox	31
page:SetTextMatrix	158	pdf.GetFindResult	31
page:SetTextRenderingMode	159	pdf.GetLastError	32
page:SetTextRise	160	pdf.GetMediaBox	33
page:SetWidth	160	pdf.GetMetaText	33
page:SetWordSpace	160	pdf.GetObjectType	34
page:SetZoom	161	pdf.GetPageLabel	35
page:ShowText	161	pdf.GetPageLen	35
page:ShowTextNextLine	161	pdf.GetPageLinks	36
page:Stroke	162	pdf.GetRects	37
page:TextOut	162	pdf.GetText	37
page:TextRect	163	pdf.GetVersion	38
page:TextWidth	163	pdf.IsPDF	38
pdf.CloseDocument	21	pdf.LoadPage	39
pdf.CreateDocument	21	pdf.OpenDocument	39
pdf.DeviceToPage	21	pdf.PageToDevice	40