

libhpdfbtbl

Generated on Sun May 8 2022 10:30:31 for libhpdfbtbl by Doxygen 1.9.3

Sun May 8 2022 10:30:31



<b>1 Introduction to libpdfbl</b>	<b>1</b>
1.1 What is this?	1
1.2 Features	1
1.3 Some Examples	2
1.3.1 Example 1 - Plain table with cell labels	2
1.3.2 Example 2 - Table with cell labels	2
1.3.3 Example 2 - Plain table with row/column spanning and table title	2
1.3.4 Example 3 - Table with labels and cell widgets	3
<b>2 Building the library</b>	<b>5</b>
2.1 The short version; TL; DR	5
2.1.1 Compiling the tar ball	5
2.2 Pre-requisites	5
2.2.1 Different versions of iconv on OSX	6
2.2.2 OSX native libiconv	6
2.2.3 OSX GNU port of libiconv	6
2.2.4 Troubleshooting OSX <code>&lt;tt&gt;libiconv&lt;/tt&gt;</code>	6
2.3 Building the library from source	7
2.3.1 Rebuilding using an existing build environment	7
2.3.2 Rebuilding from a cloned repo	7
2.4 Miscellaneous	8
2.4.1 Some notes on Compiling for debugging	8
2.4.2 Some notes on updating the documentation	9
2.4.3 Some notes on Windows build	9
2.4.4 Some notes on using C or C++ to build	9
<b>3 Getting started</b>	<b>11</b>
3.1 Creating a PDF page infrastructure	11
3.2 Your first table	12
3.3 Your second table - disconnecting program structure from data	13
3.4 Adding a header row	14
3.5 Using labels in the table cells	15
3.6 Adding a table title	16
3.7 Adjusting fonts and colors	16
<b>4 Adjusting the layout of the table</b>	<b>17</b>
4.1 Cell and row spanning	17
4.2 Adjusting column width	17
<b>5 Content and label callbacks</b>	<b>19</b>
5.1 Introducing content callback functions	19
5.2 A content callback example	20
<b>6 Error handling</b>	<b>23</b>

6.1 Translating HPDF error codes . . . . .	24
6.2 Example of setting up error handler . . . . .	24
<b>7 Style and font setting</b>	<b>25</b>
7.1 Adjusting fonts and colors . . . . .	25
7.2 Using style callbacks . . . . .	26
7.2.1 Style callback example . . . . .	27
7.3 Using style themes . . . . .	28
7.4 Adjusting grid line styles . . . . .	29
7.5 Adding zebra lines in a table . . . . .	30
<b>8 Tables layout from data</b>	<b>33</b>
8.1 Defining a table in data . . . . .	33
8.2 A first example of defining table as data . . . . .	34
8.3 A second example of defining a table as data . . . . .	34
<b>9 Widgets</b>	<b>37</b>
9.1 Overview . . . . .	37
9.1.1 1. Segmented horizontal bar example . . . . .	37
9.1.2 2. Horizontal bar example . . . . .	37
9.1.3 3. Signal strength meter example . . . . .	37
9.1.4 4. Radio sliding button example . . . . .	38
9.1.5 5. Boxed letters example . . . . .	38
9.2 Widget functions . . . . .	38
9.2.1 Segmented horizontal bar defining function . . . . .	38
9.2.2 Horizontal bar defining function . . . . .	38
9.2.3 Signal strength defining function . . . . .	39
9.2.4 Radio sliding button defining function . . . . .	39
9.2.5 Boxed letters defining function . . . . .	39
9.3 Usage . . . . .	39
<b>10 HPDFTBL API Overview</b>	<b>41</b>
10.1 Table creation related functions . . . . .	41
10.2 Table error handling . . . . .	41
10.3 Theme handling methods . . . . .	41
10.4 Table layout adjusting functions . . . . .	42
10.5 Table style modifying functions . . . . .	42
10.6 Content handling . . . . .	43
10.7 Callback handling . . . . .	43
10.8 Text encoding . . . . .	43
10.9 Misc utility function . . . . .	43
<b>11 Todo List</b>	<b>45</b>

<b>12 Data Structure Index</b>	<b>47</b>
12.1 Data Structures	47
<b>13 File Index</b>	<b>49</b>
13.1 File List	49
<b>14 Data Structure Documentation</b>	<b>51</b>
14.1 grid_style Struct Reference	51
14.1.1 Detailed Description	51
14.1.2 Field Documentation	51
14.1.2.1 color	51
14.1.2.2 line_dashstyle	52
14.1.2.3 width	52
14.2 hpdfctl Struct Reference	52
14.2.1 Detailed Description	53
14.2.2 Field Documentation	53
14.2.2.1 bottom_vmargin_factor	53
14.2.2.2 canvas_cb	54
14.2.2.3 cells	54
14.2.2.4 col_width_percent	54
14.2.2.5 cols	54
14.2.2.6 content_cb	54
14.2.2.7 content_style	55
14.2.2.8 content_style_cb	55
14.2.2.9 header_style	55
14.2.2.10 height	55
14.2.2.11 inner_hgrid	55
14.2.2.12 inner_tgrid	56
14.2.2.13 inner_vgrid	56
14.2.2.14 label_cb	56
14.2.2.15 label_style	56
14.2.2.16 minheight	56
14.2.2.17 outer_grid	57
14.2.2.18 pdf_doc	57
14.2.2.19 pdf_page	57
14.2.2.20 posx	57
14.2.2.21 posy	57
14.2.2.22 rows	57
14.2.2.23 tag	58
14.2.2.24 title_style	58
14.2.2.25 title_txt	58
14.2.2.26 use_cell_labels	58
14.2.2.27 use_header_row	58

14.2.2.28 use_label_grid_style . . . . .	59
14.2.2.29 use_zebra . . . . .	59
14.2.2.30 width . . . . .	59
14.2.2.31 zebra_color1 . . . . .	59
14.2.2.32 zebra_color2 . . . . .	60
14.2.2.33 zebra_phase . . . . .	60
14.3 hpdfdbl_cell Struct Reference . . . . .	60
14.3.1 Detailed Description . . . . .	61
14.3.2 Field Documentation . . . . .	61
14.3.2.1 canvas_cb . . . . .	61
14.3.2.2 colspan . . . . .	61
14.3.2.3 content . . . . .	61
14.3.2.4 content_cb . . . . .	61
14.3.2.5 content_style . . . . .	61
14.3.2.6 delta_x . . . . .	62
14.3.2.7 delta_y . . . . .	62
14.3.2.8 height . . . . .	62
14.3.2.9 label . . . . .	62
14.3.2.10 label_cb . . . . .	62
14.3.2.11 parent_cell . . . . .	62
14.3.2.12 rowspan . . . . .	63
14.3.2.13 style_cb . . . . .	63
14.3.2.14 textwidth . . . . .	63
14.3.2.15 width . . . . .	63
14.4 hpdfdbl_cell_spec Struct Reference . . . . .	63
14.4.1 Detailed Description . . . . .	64
14.4.2 Field Documentation . . . . .	64
14.4.2.1 canvas_cb . . . . .	64
14.4.2.2 col . . . . .	64
14.4.2.3 colspan . . . . .	64
14.4.2.4 content_cb . . . . .	65
14.4.2.5 label . . . . .	65
14.4.2.6 label_cb . . . . .	65
14.4.2.7 row . . . . .	65
14.4.2.8 rowspan . . . . .	65
14.4.2.9 style_cb . . . . .	66
14.5 hpdfdbl_errcode_entry Struct Reference . . . . .	66
14.5.1 Detailed Description . . . . .	66
14.5.2 Field Documentation . . . . .	66
14.5.2.1 errcode . . . . .	66
14.5.2.2 errstr . . . . .	66
14.6 hpdfdbl_spec Struct Reference . . . . .	67

14.6.1 Detailed Description	67
14.6.2 Field Documentation	67
14.6.2.1 cell_spec	67
14.6.2.2 cols	68
14.6.2.3 content_cb	68
14.6.2.4 height	68
14.6.2.5 label_cb	68
14.6.2.6 post_cb	68
14.6.2.7 rows	69
14.6.2.8 style_cb	69
14.6.2.9 title	69
14.6.2.10 use_header	69
14.6.2.11 use_labelgrid	69
14.6.2.12 use_labels	70
14.6.2.13 width	70
14.6.2.14 xpos	70
14.6.2.15 ypos	70
14.7 hpdftbl_theme Struct Reference	70
14.7.1 Detailed Description	71
14.7.2 Field Documentation	71
14.7.2.1 bottom_vmargin_factor	71
14.7.2.2 content_style	71
14.7.2.3 header_style	72
14.7.2.4 inner_hborder	72
14.7.2.5 inner_tborder	72
14.7.2.6 inner_vborder	72
14.7.2.7 label_style	72
14.7.2.8 outer_border	73
14.7.2.9 title_style	73
14.7.2.10 use_header_row	73
14.7.2.11 use_label_grid_style	73
14.7.2.12 use_labels	73
14.7.2.13 use_zebra	74
14.7.2.14 zebra_color1	74
14.7.2.15 zebra_color2	74
14.7.2.16 zebra_phase	74
14.8 line_dash_style Struct Reference	74
14.8.1 Detailed Description	75
14.8.2 Field Documentation	75
14.8.2.1 dash_ptn	75
14.8.2.2 num	75
14.9 text_style Struct Reference	75

14.9.1 Detailed Description	76
14.9.2 Field Documentation	76
14.9.2.1 background	76
14.9.2.2 color	76
14.9.2.3 font	77
14.9.2.4 fsize	77
14.9.2.5 halign	77
<b>15 File Documentation</b>	<b>79</b>
15.1 bootstrap.sh File Reference	79
15.1.1 Detailed Description	79
15.2 dbgblld.sh File Reference	79
15.2.1 Detailed Description	80
15.3 docupload.sh.in File Reference	80
15.3.1 Detailed Description	81
15.3.2 Variable Documentation	81
15.3.2.1 GITHUB_USER	81
15.3.2.2 PDFFILE_COPY	81
15.4 stdbld.sh File Reference	81
15.4.1 Detailed Description	82
15.5 config.h	82
15.6 hpdfdbl.c File Reference	83
15.6.1 Detailed Description	86
15.6.2 Function Documentation	87
15.6.2.1 HPDF_RoundedCornerRectangle()	87
15.6.2.2 hpdfdbl_clear_spanning()	88
15.6.2.3 hpdfdbl_create()	88
15.6.2.4 hpdfdbl_create_title()	89
15.6.2.5 hpdfdbl_default_table_error_handler()	89
15.6.2.6 hpdfdbl_destroy()	90
15.6.2.7 hpdfdbl_encoding_text_out()	90
15.6.2.8 hpdfdbl_get_anchor_top_left()	91
15.6.2.9 hpdfdbl_get_errstr()	91
15.6.2.10 hpdfdbl_get_last_auto_height()	92
15.6.2.11 hpdfdbl_get_last_errcode()	92
15.6.2.12 hpdfdbl_set_anchor_top_left()	93
15.6.2.13 hpdfdbl_set_background()	93
15.6.2.14 hpdfdbl_set_bottom_vmargin_factor()	93
15.6.2.15 hpdfdbl_set_canvas_cb()	94
15.6.2.16 hpdfdbl_set_cell()	94
15.6.2.17 hpdfdbl_set_cell_canvas_cb()	95
15.6.2.18 hpdfdbl_set_cell_content_cb()	96



15.6.2.19	hpdfctl_set_cell_content_style()	96
15.6.2.20	hpdfctl_set_cell_content_style_cb()	97
15.6.2.21	hpdfctl_set_cell_label_cb()	98
15.6.2.22	hpdfctl_set_cellspan()	98
15.6.2.23	hpdfctl_set_col_content_style()	99
15.6.2.24	hpdfctl_set_colwidth_percent()	100
15.6.2.25	hpdfctl_set_content()	100
15.6.2.26	hpdfctl_set_content_cb()	101
15.6.2.27	hpdfctl_set_content_style()	102
15.6.2.28	hpdfctl_set_content_style_cb()	102
15.6.2.29	hpdfctl_set_errhandler()	103
15.6.2.30	hpdfctl_set_header_halign()	104
15.6.2.31	hpdfctl_set_header_style()	104
15.6.2.32	hpdfctl_set_inner_grid_style()	105
15.6.2.33	hpdfctl_set_inner_hgrid_style()	105
15.6.2.34	hpdfctl_set_inner_tgrid_style()	106
15.6.2.35	hpdfctl_set_inner_vgrid_style()	107
15.6.2.36	hpdfctl_set_label_cb()	107
15.6.2.37	hpdfctl_set_label_style()	108
15.6.2.38	hpdfctl_set_labels()	108
15.6.2.39	hpdfctl_set_line_dash()	109
15.6.2.40	hpdfctl_set_min_rowheight()	110
15.6.2.41	hpdfctl_set_outer_grid_style()	110
15.6.2.42	hpdfctl_set_row_content_style()	111
15.6.2.43	hpdfctl_set_tag()	112
15.6.2.44	hpdfctl_set_text_encoding()	112
15.6.2.45	hpdfctl_set_title()	112
15.6.2.46	hpdfctl_set_title_halign()	114
15.6.2.47	hpdfctl_set_title_style()	115
15.6.2.48	hpdfctl_set_zebra()	115
15.6.2.49	hpdfctl_set_zebra_color()	116
15.6.2.50	hpdfctl_stroke()	116
15.6.2.51	hpdfctl_stroke_from_data()	117
15.6.2.52	hpdfctl_use_header()	118
15.6.2.53	hpdfctl_use_labelgrid()	118
15.6.2.54	hpdfctl_use_labels()	119
15.7	hpdfctl.h File Reference	120
15.7.1	Detailed Description	126
15.7.2	Macro Definition Documentation	126
15.7.2.1	_HPDFTBL_SET_ERR	126
15.7.2.2	DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR	127
15.7.2.3	hpdfctl_cm2dpi	127

15.7.3 Typedef Documentation	127
15.7.3.1 hpdf_text_style_t	127
15.7.3.2 hpdftbl_callback_t	128
15.7.3.3 hpdftbl_canvas_callback_t	128
15.7.3.4 hpdftbl_cell_spec_t	128
15.7.3.5 hpdftbl_cell_t	128
15.7.3.6 hpdftbl_content_callback_t	129
15.7.3.7 hpdftbl_content_style_callback_t	129
15.7.3.8 hpdftbl_error_handler_t	129
15.7.3.9 hpdftbl_grid_style_t	129
15.7.3.10 hpdftbl_line_dashstyle_t	130
15.7.3.11 hpdftbl_spec_t	130
15.7.3.12 hpdftbl_t	130
15.7.3.13 hpdftbl_text_align_t	130
15.7.3.14 hpdftbl_theme_t	131
15.7.4 Enumeration Type Documentation	131
15.7.4.1 hpdftbl_dashstyle	131
15.7.4.2 hpdftbl_text_align	131
15.7.5 Function Documentation	132
15.7.5.1 HPDF_RoundedCornerRectangle()	132
15.7.5.2 hpdftbl_apply_theme()	132
15.7.5.3 hpdftbl_clear_spanning()	133
15.7.5.4 hpdftbl_create()	133
15.7.5.5 hpdftbl_create_title()	134
15.7.5.6 hpdftbl_default_table_error_handler()	134
15.7.5.7 hpdftbl_destroy()	135
15.7.5.8 hpdftbl_destroy_theme()	135
15.7.5.9 hpdftbl_encoding_text_out()	136
15.7.5.10 hpdftbl_get_anchor_top_left()	136
15.7.5.11 hpdftbl_get_default_theme()	137
15.7.5.12 hpdftbl_get_errstr()	137
15.7.5.13 hpdftbl_get_last_auto_height()	138
15.7.5.14 hpdftbl_get_last_errcode()	138
15.7.5.15 hpdftbl_hpdf_get_errstr()	138
15.7.5.16 hpdftbl_set_anchor_top_left()	139
15.7.5.17 hpdftbl_set_background()	139
15.7.5.18 hpdftbl_set_bottom_vmargin_factor()	140
15.7.5.19 hpdftbl_set_canvas_cb()	140
15.7.5.20 hpdftbl_set_cell()	141
15.7.5.21 hpdftbl_set_cell_canvas_cb()	141
15.7.5.22 hpdftbl_set_cell_content_cb()	142
15.7.5.23 hpdftbl_set_cell_content_style()	143

15.7.5.24	hpdfctl_set_cell_content_style_cb()	143
15.7.5.25	hpdfctl_set_cell_label_cb()	144
15.7.5.26	hpdfctl_set_cellspan()	145
15.7.5.27	hpdfctl_set_col_content_style()	145
15.7.5.28	hpdfctl_set_colwidth_percent()	146
15.7.5.29	hpdfctl_set_content()	146
15.7.5.30	hpdfctl_set_content_cb()	147
15.7.5.31	hpdfctl_set_content_style()	148
15.7.5.32	hpdfctl_set_content_style_cb()	148
15.7.5.33	hpdfctl_set_errhandler()	149
15.7.5.34	hpdfctl_set_header_halign()	149
15.7.5.35	hpdfctl_set_header_style()	150
15.7.5.36	hpdfctl_set_inner_grid_style()	150
15.7.5.37	hpdfctl_set_inner_hgrid_style()	151
15.7.5.38	hpdfctl_set_inner_tgrid_style()	152
15.7.5.39	hpdfctl_set_inner_vgrid_style()	152
15.7.5.40	hpdfctl_set_label_cb()	153
15.7.5.41	hpdfctl_set_label_style()	153
15.7.5.42	hpdfctl_set_labels()	154
15.7.5.43	hpdfctl_set_min_rowheight()	155
15.7.5.44	hpdfctl_set_outer_grid_style()	155
15.7.5.45	hpdfctl_set_row_content_style()	156
15.7.5.46	hpdfctl_set_tag()	156
15.7.5.47	hpdfctl_set_text_encoding()	157
15.7.5.48	hpdfctl_set_title()	157
15.7.5.49	hpdfctl_set_title_halign()	158
15.7.5.50	hpdfctl_set_title_style()	158
15.7.5.51	hpdfctl_set_zebra()	159
15.7.5.52	hpdfctl_set_zebra_color()	159
15.7.5.53	hpdfctl_stroke()	160
15.7.5.54	hpdfctl_stroke_from_data()	161
15.7.5.55	hpdfctl_stroke_grid()	161
15.7.5.56	hpdfctl_table_widget_letter_buttons()	162
15.7.5.57	hpdfctl_use_header()	162
15.7.5.58	hpdfctl_use_labelgrid()	164
15.7.5.59	hpdfctl_use_labels()	165
15.7.5.60	hpdfctl_widget_hbar()	166
15.7.5.61	hpdfctl_widget_segment_hbar()	167
15.7.5.62	hpdfctl_widget_slide_button()	167
15.7.5.63	hpdfctl_widget_strength_meter()	168
15.8	hpdfctl.h	169
15.9	hpdfctl_errstr.c File Reference	175

15.9.1 Detailed Description	175
15.9.2 Function Documentation	176
15.9.2.1 hpdfctl_hpdl_get_errstr()	176
15.10 hpdfctl_grid.c File Reference	176
15.10.1 Detailed Description	176
15.10.2 Function Documentation	176
15.10.2.1 hpdfctl_stroke_grid()	177
15.11 hpdfctl_theme.c File Reference	177
15.11.1 Detailed Description	178
15.11.2 Macro Definition Documentation	178
15.11.2.1 HPDFTBL_DEFAULT_CONTENT_STYLE	178
15.11.2.2 HPDFTBL_DEFAULT_HEADER_STYLE	179
15.11.2.3 HPDFTBL_DEFAULT_INNER_HGRID_STYLE	179
15.11.2.4 HPDFTBL_DEFAULT_INNER_VGRID_STYLE	179
15.11.2.5 HPDFTBL_DEFAULT_LABEL_STYLE	179
15.11.2.6 HPDFTBL_DEFAULT_OUTER_GRID_STYLE	180
15.11.2.7 HPDFTBL_DEFAULT_ZEBRA_COLOR1	180
15.11.2.8 HPDFTBL_DEFAULT_ZEBRA_COLOR2	180
15.11.3 Function Documentation	180
15.11.3.1 hpdfctl_apply_theme()	180
15.11.3.2 hpdfctl_destroy_theme()	181
15.11.3.3 hpdfctl_get_default_theme()	181
15.12 hpdfctl_widget.c File Reference	182
15.12.1 Detailed Description	183
15.12.2 Macro Definition Documentation	183
15.12.2.1 FALSE	183
15.12.2.2 TRUE	183
15.12.3 Function Documentation	183
15.12.3.1 hpdfctl_table_widget_letter_buttons()	183
15.12.3.2 hpdfctl_widget_hbar()	184
15.12.3.3 hpdfctl_widget_segment_hbar()	185
15.12.3.4 hpdfctl_widget_slide_button()	185
15.12.3.5 hpdfctl_widget_strength_meter()	186
<b>16 Example Documentation</b>	<b>187</b>
16.1 example01.c	187
16.2 tut_ex01.c	193
16.3 tut_ex02.c	194
16.4 tut_ex02_1.c	195
16.5 tut_ex03.c	197
16.6 tut_ex04.c	198
16.7 tut_ex05.c	200

---

16.8 tut_ex06.c . . . . .	201
16.9 tut_ex07.c . . . . .	203
16.10 tut_ex08.c . . . . .	205
16.11 tut_ex09.c . . . . .	207
16.12 tut_ex10.c . . . . .	209
16.13 tut_ex11.c . . . . .	211
16.14 tut_ex12.c . . . . .	212
16.15 tut_ex13_1.c . . . . .	214
16.16 tut_ex13_2.c . . . . .	215
16.17 tut_ex14.c . . . . .	217
16.18 tut_ex15.c . . . . .	220
16.19 tut_ex15_1.c . . . . .	221
16.20 tut_ex20.c . . . . .	223
<b>Index</b>	<b>225</b>



# Chapter 1

## Introduction to hpdftbl

### 1.1 What is this?

The Haru PDF library is a great way to programmatically produce PDFs from programs. However, in many instances the best way to present data produced is as a grid (or table). To manually create and setup such tables in the Haru PDF library is of course possible but only painstakingly so.

This C/C++ library `libhpdftbl` will facilitate the creation of tables with the Haru PDF library as well as handling the pesky issue of character conversion needed between UTF-8 and the internal standard used by PDF and Lib Haru. In addition to mere normal table the library also supports the creation of forms where each cell has a label similar to "formal" paper forms. This is a great way to present structured data from a DB.

This library provides a flexible abstraction for creating advanced tables with a model-view-controller like setup. This allows an easy way to separate the layout of the table from the actual data in the table.

### 1.2 Features

- Supports both C/C++
- Supports both OSX/Linux builds and their different dynamic library variants
- Fully supports UTF-8 with automatic conversion to PDF character encoding
- Supports multiple paradigms for creating and populating tables
  - Directly store value in table cell
  - Create a data structure (2D-Array) with all data to be set at once
  - Use callback populating functions with identifying tags for each table cell
- Options to use labels in table cell to create forms
- Support for predefined widgets in table cell to illustrate values
- Complete control of background color, fonts, and frame colors
- Possible to use table themes that provide pre-defined look-and-feel for table
- Both dynamic and static library provided
- Last but not least; extensive documentation and almost guaranteed to be bug free after being tested in production for over 7 years!

## 1.3 Some Examples

### Note

All code examples can be found in the `examples/` directory or in the [Examples](#) section of this manual. All examples will be explained in this manual.

### 1.3.1 Example 1 - Plain table with cell labels

[tut\\_ex02\\_1.c](#)

Header 0	Header 1	Header 2	Header 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15

### 1.3.2 Example 2 - Table with cell labels

[example01.c](#)

<i>Label 0:</i> Content 0	<i>Label 1:</i> Content 1	<i>Label 2:</i> Content 2	<i>Label 3:</i> Content 3
<i>Label 4:</i> Content 4	<i>Label 5:</i> Content 5	<i>Label 6:</i> Content 6	<i>Label 7:</i> Content 7
<i>Label 8:</i> Content 8	<i>Label 9:</i> Content 9	<i>Label 10:</i> Content 10	<i>Label 11:</i> Content 11
<i>Label 12:</i> Content 12	<i>Label 13:</i> Content 13	<i>Label 14:</i> Content 14	<i>Label 15:</i> Content 15
<i>Label 16:</i> Content 16	<i>Label 17:</i> Content 17	<i>Label 18:</i> Content 18	<i>Label 19:</i> Content 19

### 1.3.3 Example 2 - Plain table with row/column spanning and table title







[example01.c](#)



Example 3: Table cell spannings and full grid and header			
Content 0	Content 1		
Label 4: Content 4	Label 5: Content 5		
Label 8: Content 8	Label 9: Content 9	Label 10: Content 10	
Label 12: Content 12	Label 13: Content 13	Label 14: Content 14	Label 15: Content 15
Label 16: Content 16	Label 17: Content 17		
Label 20: Content 20			
Label 24: Content 24	Label 25: Content 25	Label 26: Content 26	Label 27: Content 27
Label 28: Content 28	Label 29: Content 29	Label 30: Content 30	
Label 32: Content 32	Label 33: Content 33		

### 1.3.4 Example 3 - Table with labels and cell widgets

[example01.c](#)

Example 5: Using widgets in cells			
Horizontal seg bar:  40%	Label 1: Content 1	Label 2: Content 2	Label 3: Content 3
Horizontal bar:  60%	Label 5: Content 5	Label 6: Content 6	Label 7: Content 7
Slider on: 	Label 9: Content 9	Label 10: Content 10	Label 11: Content 11
Slider off: 	Label 13: Content 13	Label 14: Content 14	Label 15: Content 15
Strength meter: 	Label 17: Content 17	Label 18: Content 18	Label 19: Content 19
Boxed letters: 	Label 21: Content 21	Label 22: Content 22	Label 23: Content 23



## Chapter 2

# Building the library

## 2.1 The short version; TL; DR

### 2.1.1 Compiling the tar ball

The tar-ball should be trivial to build and install if the necessary [pre-requisites](#) are fulfilled.

```
$ tar xzf libhpdf-1.0.0.tar.gz
$ cd libhpdf-1.0.0
$ ./configure && make
$ make install
```

If any libraries are missing the `configure` process will discover this and tell what needs to be installed. If successfully, the above commands will compile and install the library in `/usr/local` subtree. It will build and install both a static and dynamic library

#### Note

By calling `./configure -h` a list of possible options on how the library should be compiled and installed will be shown.

Depending on the system there might also be pre-built binary packages available for install directly via `apt` on Linux or `brew` on OSX.

## 2.2 Pre-requisites

#### Note

OSX Package manager: We recommend using `brew` as the package manager for OSX.

There are two external libraries required to rebuild `libhpdf` and more importantly use the library with an actual application and these are:

1. **libhpdf** - The Haru PDF library. On OSX this is most easily installed by using the `brew` OSX package manager. The library is available as `libharu` as of this writing the latest version is `libharu-2.3.0`
2. **iconv** - The character encoding conversion library. On OSX > 11.x this is included by default once you have `xcode` command line tools installed which is basically a pre-requisite required for all development on OSX. \*(On ancient versions of OSX this was not the case.)\*

## 2.2.1 Different versions of iconv on OSX

Unfortunately there are two main versions of `libiconv` readily available for OSX which are incompatible as one uses the prefix `"iconv_"` and the other `"libiconv_"` on its exported functions. Compiling `libhpdf.tbl` requires the first of these which is the prevalent version and the default on both OSX and Linux.

This is almost exclusively an issue for those that actively develop on OSX and may have over time installed multiple versions of libraries and as such are aware of these challenges.

## 2.2.2 OSX native libiconv

After installing `xcode` command line tools on OSX you can assume that a library called `/usr/lib/iconv.dylib` is available. However, if you actually try to list this library in `/usr/lib` you will not find it! Still, if you link your code with `-liconv` it will work as expected. How come?

The reason is the way OSX handles different library versions for different OSX SDKs. Since `xcode` supports developing for different OSX versions the SDK would need to include a complete setup of all `*.dylib` of the right version for each included version of the SDK. To reduce disk space all dynamic libraries are rolled-up in a dynamic link shared cache for each SDK version. The tool chain (e.g. `gcc`) have been augmented to be aware of this. Hence, there is no need to have libraries in `/usr/lib`. Instead, OSX from v11 and onwards uses the concept of *stub libraries* `*.tbd` (tbd stands for "text based description") which are much smaller text files with some meta information about the library used by the tool-chain.

For example for SDK 12.3 the stub for `libiconv` can be found at  
`/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/lib/libiconv.tbd`

and the corresponding include header at  
`/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/iconv.h`

## 2.2.3 OSX GNU port of libiconv

If you have happened to install `libiconv` via the MacPorts you are out of luck and need to change. MacPorts uses the GNU version which uses the prefix `"libiconv_"` for its exported function and is not compatible since the table library assumes the naming convention of the standard OSX version (after v11)

## 2.2.4 Troubleshooting OSX `libiconv`

1. Find out all installed versions of `libiconv` on your installation

```
$> find / -iregex '.*libiconv.*' 2> /dev/null
```

The `"2> /dev/null"` makes sure you don't get a lot of noise "permission denied"

2. Find out the SDK path that is actively used

```
$> xcrun --show-sdk-path
```

3. Check you `PATH` variable

```
$> echo $PATH
```

## 2.3 Building the library from source

There are two levels of rebuilding the library

1. Using a build environment to rebuild the library
2. Rebuilding from a cloned repo and rebuild the build environment

### 2.3.1 Rebuilding using an existing build environment

Rebuilding the library using a pre-configured build environment only requires `gcc` and `make` together with the standard C/C++ libraries to be installed.

The library source with suitable build-environment is distributed as a tar-ball

1. `libhpdf-tbl-x.y.z.tar.gz`

This tar-ball includes a build environment constructed with the GNU autotools. This means that after downloading the tar-ball you can rebuild the library as so:

```
$ tar xzf libhpdf-tbl-1.0.0.tar.gz
$ cd libhpdf-1.0.0
$ ./configure && make
... (output from the configuration and build omitted) ...
```

and then (optionally) install the library with

```
$ make install
```

By default, the library will install under the `/usr/local` but that can be adjusted by using the `--prefix` parameter to `configure`. For example

```
$ tar xzf libhpdf-tbl-1.0.0.tar.gz
$ cd libhpdf-1.0.0
$ ./configure --prefix=/usr && make
... (output from the configuration and build omitted) ...
```

Please refer to `configure -h` for other possible configurations.

### 2.3.2 Rebuilding from a cloned repo

#### Note

This is for experienced developers!

The repo does not include any of the generated files as the tar-ball does. This means that the following build tools need to be setup in order to fully rebuild from a cloned repo.

1. A complete set of GNU compiler chain (or on OSX clang)
2. the `GNU autotools` (autoconf, automake, libtool)
3. `Doxygen` in order to rebuild the documentation

Since it is completely out of the scope to describe the intricacies of the GNU autotools we will only show what to do assuming this tool chain have already been installed.

To simplify the potentially painful (?) bootstrap of creating a full autotools environment from the cloned repo a utility script that does this is provided in the form of [scripts/bootstrap.sh](#). After cloning the repo run (from the `libhpdfctl` directory)

```
./scripts/bootstrap.sh
```

This script will now run `autoreconf`, `automake`, `glibtoolize` as needed in order to create a full build environment. It will also run `configure` and if everything works as expected the last lines you will see (on OSX) will be

```
...
config.status: executing libtool commands
configure: -----
configure: INSTALLATION SUMMARY:
configure:   - Build configured for OSX.
configure:   - Can rebuild HTML docs with Doxygen.
configure:   - Can also create PDF docs (have LaTeX).
configure:   - Installing to /usr/local
configure: -----
```

and then to compile the library

```
$> make
```

The simplest way to verify that everything works is to run the built-in unit/integration tests

```
$> make check
Info: PASS: tut_ex01
Info: PASS: tut_ex02
<omitted ...>
Info: PASS: tut_ex20
Info: =====
Info: PASS!
Info: =====
```

To then install the library

```
$> make install
```

This will install headers and library under `"/usr/local"` (unless the prefix was changed when running the `configure`)

## 2.4 Miscellaneous

### 2.4.1 Some notes on Compiling for debugging

Since the library builds with `libtool` and this tool will generate a wrapper shell script for each example to load the, not yet installed, library it also means this "executable" shell script cannot directly be used to debug with for example `gdb`.

The solution for this is to configure the library to only build static libraries which are directly linked with the example binaries and as such can be debugged as usual. It is also a good idea to disable optimization during debugging to make the source better follow the execution while stepping through the code. This configuration is done with:

```
$> ./configure --disable-shared CFLAGS="-O0 -ggdb"
```

After this all the examples will be statically linked and can be debugged as usual

An alternative way (as recommended in the [libtool manual](#)) is to launch the debugger with:

```
$> libtool --mode=execute gdb <example program>
```

As a convenience a script is provided to handle the debug build configuration [scripts/dbgbld.sh](#)

### 2.4.2 Some notes on updating the documentation

By design the documentation is not updated by the default make target in order to minimize the build time during development. To rebuild the *html* documentation build the target

```
$> make html
```

and to rebuild the *PDF* version build the target

```
$> make pdf
```

The resulting documentations are stored under `docs/out/html` and `docs/out/latex/refman.pdf`

#### Warning

There is a shell script `scripts/docupload.sh.in` that the author (i.e. me!) uses to upload the HTML and PDF documentation to the GitHub pages of the author. For obvious reason this script will not work for anyone else since it requires write access to the doc repo (through an SSL certificate).

### 2.4.3 Some notes on Windows build

The source files are suitable augmented to also compile on MS Windows with selective defines. However, since I have no longer access to a Windows system to verify the workings this is left as an exercise to the reader. Hence, this should be considered as the best effort.

### 2.4.4 Some notes on using C or C++ to build

The source files are also suitable augmented to compile on both a C and a C++ compiler. However, the default build environment is set up for a pure C library build. To add a configuration switch for this would be the sensible way to handle this. This is not done and again, is left as an exercise for the reader.





## Chapter 3

# Getting started

In this section we will introduce the basic usage of the `hpdftbl` library. We will start simple and work us all the way to complex tables and explain what is happening as we go along.

We will not assume any knowledge of the table library, but **we will assume that you are familiar with the plain Haru PDF library**.

### 3.1 Creating a PDF page infrastructure

Before we start creating a table we need to set up a plain PDF page with the core HPDF library. The HPDF library has excellent documentation on how to do this, and we will use the same simple setup for all our examples. We will create a document in A4 size that have one page. For this we use a few utility functions and our `main()` will always have the following structure:

```
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, TRUE);
    create_table_<NAME_OF_EXAMPLE>(pdf_doc, pdf_page);

    stroke_pdfdoc(pdf_doc, OUTPUT_FILE);
    return EXIT_SUCCESS;
}
```

In the `examples` directory the full source code for the setup and stroke function can be found in all the tutorial examples, for example [tut\\_ex01.c](#). They are very basic and follows the standard hpdf library methodology. The `setup_hpdf()` creates a new document and one A4 page and the `stroke_pdfdoc()` strokes the document to the given output file.

In the following we will focus only on the `create_table_<NAME_OF_EXAMPLE>()` function which will use the two parameters `pdf_doc` and `pdf_page` to refer to the document and page to construct the table.

#### Note

In order to make the examples robust and compatible with both Windows and Linux/OSX systems some conditional compilation instructions are also used, but we will not display them while discussing the basic usage to keep the focus on what matters.

The full source for all example are available in the `examples/` directory as well as in the Examples section of this manual.

## 3.2 Your first table

[tut\\_ex01.c](#)

The first example shows the absolute most basic usage. We create a 2x2 table in steps as follows

First we construct a table handle for a 2x2 table

```
const size_t num_rows = 2;
const size_t num_cols = 2;
hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
```

Here we note that:

- The size of the table has to be determined before the table handle is created
- All table function will refer to this handle, and we will always use the variable name `tbl` for this handle
- We use `size_t` instead of `int` since the table dimension is a size and as such can never be negative. In C it is always good practice to use `size_t` for positive numeric entities.

Once we have the table handle we can start to add content in these cells. For now lets just put a string that indicates the cells position.

```
hpdf_tbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
hpdf_tbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
hpdf_tbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
hpdf_tbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
```

### Note

You can ignore the `NULL` argument for now (it will be explained shortly).

Here we note that:

- Cells are referred to starting from the top left cell that is cell (0x0)

Now It's time to size and position the table on the page. As a minimum you must specify the `x` and `y` position as well as the width of the table. The library is smart enough to automatically figure out the height (but it is also possible to force a larger height than strictly necessary)

The native coordinate system for PDF pages are given as the printing unit of DPI or *dots per inch*. By default, the resolution of a PDF is 72 DPI.

To make it easier to directly set the size and position in centimeters a convenience function `hpdf_tbl_cm2dpi()` can be used.

### Note

For precision positioning it is more accurate to give the position and sizes in dots directly.

In this example we set the size and position in centimeters. We position the top left of the table *1cm* below and *1cm* to the right of the top left corner of the paper and make the table *5cm* wide as follows:

```
HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
HPDF_REAL width = hpdf_tbl_cm2dpi(5);
HPDF_REAL height = 0; // Calculate height automatically
```

Now, there are several important observations to be made here:

- The origin of the paper coordinate system is bottom left which is (0,0)
- The anchor position by default is the top-left corner of the table (this can be adjusted by calling `hpdf_tbl_set_anchor_top_left(FALSE)` function which will make the bottom left the anchor point instead)
- We use a predefined constant `A4PAGE_HEIGHT_IN_CM` to position the table vertically 1 cm from the top of the paper
- We let the library calculate the minimum table height automatically (based on the font height used in the table)

Now the only thing remaining is to print or stroke the table to the page

```
hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
```

and we are done!

If we put it all together it will give us the following basic table creation code

```
void
create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;

    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    hpdf_tbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
    hpdf_tbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
    hpdf_tbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
    hpdf_tbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

The generated table is shown in **Figure 1**. ([tut\\_ex01.c](#))

Cell 0x0	Cell 0x1
Cell 1x0	Cell 1x1

**Figure 1:** Your first table.

As we explained above the coordinate system is in postscript dots. For precision positioning it might be useful to visualize this grid on the page. By using the `hpdf_tbl_stroke_grid()` function such a grid can be displayed on a page to help with positioning. If we add the grid to the page and show the upper left area of the paper with the grid we can view its positioning in the grid as shown in **Figure 2**.

Cell 0x0	Cell 0x1
Cell 1x0	Cell 1x1

**Figure 2:** Your first table in the page coordinate system showing the upper left part of the paper.

Since this is an A4 page it will have a height of roughly 841 points or 29.7cm

### 3.3 Your second table - disconnecting program structure from data

One drawback of the program in the first example above is that if we want to have a different table size we need to actually change the code since we need one function call to store the data to be displayed in each cell. Wouldn't it be better if we could just supply an array with the data we want to display?

The function to do just that is

```
hpdf_tbl_set_content(hpdf_tbl_t tbl, char **content)
```

The content data is a 1-dimensional array of string pointers. Where each row is consecutive in the array. For example to create dummy data indicating what array position goes into what cell you could use the following setup:

```
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
```

**Note**

We allocate each string dynamically in the dummy-data and since the program is just an illustration and terminates after the page has been created we never bother to free this memory. In a real life scenario this would of course be crucial!

We could then augment example 01 using this more efficient way to specify data as so:

```
void
create_table_ex02(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically

    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

[tut\\_ex02.c](#)

Running the code above in our infrastructure will give

Content 0	Content 1
Content 2	Content 3

**Figure 3:** Specifying data in a table with an array of string pointers. ([tut\\_ex02.c](#))

In the above (small) example it might not have been a big save but if you have a table with 20x10 rows \* cols then you will soon appreciate this way of specifying data.

There is even one more way of specifying data that in some situations are more efficient and allows a clear division between the table structure and look&feel and its data. This more efficient way is achieved by using cell callbacks either directly in individual cells or in one go by specifying the entire table as a data structure by using the `hpdf_tbl_stroke_from_data()` function. This will be described later when we discuss how to use callback functions.

But now it is time to explain the NULL value in the first example when we specified the content with the `hpdf_tbl_set_cell()` function.

### 3.4 Adding a header row

While it is possible (as discussed in section [Style and font setting](#) and [Fonts and Colors](#) ) to manually adjust the font, size, style, background etc. on each cell individually there is a convenient shortcut to create a basic table with a header using the `hpdf_tbl_use_header()` function. By modifying the code above and add this line we get the following code and resulting table

```
void
create_table_ex11(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    hpdf_tbl_use_header(tbl, TRUE);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0; // Calculate height automatically

    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

The resulting table can be seen in **Figure 4**. We also modified the dummy data to have the word "Header" in the first row (for details see [tut\\_ex02\\_1.c](#) )

Header 0	Header 1	Header 2	Header 3
Content 0	Content 1	Content 2	Content 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15

**Figure 4:** Adding automatic header formatted row ([tut\\_ex02\\_1.c](#))

## 3.5 Using labels in the table cells

A variant of a table is to present data with a short label describing what kind of data is displayed. This is often used when a table is used to present a data form. An example of this is shown in **Figure 4.** below.

Label 1	Label 2
Label 3	Label 4

**Figure 4:** Specifying labels for each cell. ([tut\\_ex03.c](#))

Adding labels requires three things:

1. Enable the "label" feature with a call to `hpdftbl_use_labels(tbl, TRUE);`
2. Add the text that should be the label. Specifying these labels can either be done using the `hpdftbl_set_cell()` function as in

```
hpdftbl_set_cell(tbl, 0, 0, "Label 1", "Cell 0x0");
hpdftbl_set_cell(tbl, 0, 1, "Label 2", "Cell 0x1");
hpdftbl_set_cell(tbl, 1, 0, "Label 3", "Cell 1x0");
hpdftbl_set_cell(tbl, 1, 1, "Label 4", "Cell 1x1");
```

or it can be done using the analog of specifying the labels in an array using the function `hpdftbl_set_labels()`.
3. In addition, there is one more key setting and that is whether the left cell border should be the whole cell or just the table height as was shown in **Figure 4.** above. This option is specified with `hpdftbl_use_labelgrid()`.
4. By default, the left border is from top to bottom. The differences between the two variants is shown in **Figure 5.** below.

Label 1	Cell 0x0
Label 2	Cell 0x1

Label 1	Cell 0x0
Label 2	Cell 0x1

**Figure 5:** The two variants of left cell border with labels.

Except for the simplest of tables both the table content and the labels should be specified in an array.

We therefore start by amending our dummy data creation function to also create the data for the labels. It will now look like this:

```
typedef char **content_t;
void
setup_dummy_data(content_t *content, content_t *labels,
                 size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    *labels = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            snprintf(buff, sizeof(buff), "Label %zu", cnt);
            (*labels)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
```

In the same way as before we call the functions to specify both the content and the labels

```
setup_dummy_data(&content, &labels, num_rows, num_cols);
hpdftbl_set_content(tbl, content);
hpdftbl_set_labels(tbl, labels);
```

and finally we also enable labels and the short variant of the left cell border

```
hpdftbl_use_labels(tbl, TRUE);
hpdftbl_use_labelgrid(tbl, TRUE);
```

the remaining code we can leave untouched. With this we get the result shown in **Figure 4.** with the full code for the table shown below.

```
void
create_table_ex04(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
```

```

const size_t num_rows = 2;
const size_t num_cols = 2;

hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
content_t content, labels;
setup_dummy_data(&content, &labels, num_rows, num_cols);
hpdf_tbl_set_content(tbl, content);
hpdf_tbl_set_labels(tbl, labels);

hpdf_tbl_use_labels(tbl, TRUE);
hpdf_tbl_use_labelgrid(tbl, TRUE);
HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
HPDF_REAL width = hpdf_tbl_cm2dpi(5);
HPDF_REAL height = 0; // Calculate height automatically
hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

```

[tut\\_ex04.c](#)

### 3.6 Adding a table title

We have one last part of the table we haven't yet used and that is the table title. In the previous examples we created a table using `hpdf_tbl_create()` but there is also `hpdf_tbl_create_title()`. A title can also be added to an existing table (or perhaps updated) using `hpdf_tbl_set_title()`

To create a table with a title

```

char *table_title = "tut_ex05: 2x2 table";
hpdf_tbl_t tbl = hpdf_tbl_create_title(num_rows, num_cols, table_title);

```

A table title occupies the top of the table in its own row which isn't part of the counting if the normal columns.



**Figure 6:** Adding a title for the table. ([tut\\_ex05.c](#))

It is possible to adjust the colors, font-properties, and alignments of the title with two additional functions `hpdf_tbl_set_title_style()` and `hpdf_tbl_set_title_halign()`

### 3.7 Adjusting fonts and colors

The one thing we have skipped over so far and just used the defaults is the look & feel of the table as far as colors and fonts go. It is possible to adjust these setting at several levels of granularity. It is possible to:

1. Adjust the entire table in one go using `hpdf_tbl_set_content_style()`
2. Adjust one entire column using `hpdf_tbl_set_col_content_style()`
1. Adjust one entire row in using `hpdf_tbl_set_row_content_style()`
1. Adjust individual cells using `hpdf_tbl_set_content_style()`

It is also possible to adjust the color and thickness of the borders, but we will not discuss this more here and instead refer the reader to the API documentation.

#### Note

We should also mention that there is a concept of a look & feel theme for the table which can be used to adjust all the parameters at once. This is discussed in [Using themes](#).

## Chapter 4

# Adjusting the layout of the table

The table can be modified both by adjusting the width of columns and how many rows and columns a cell is spanning.

### 4.1 Cell and row spanning

A common way to modify a table is to have a cell spanning either multiple columns, multiple rows or both. This is done using the function

```
int
hpdftbl_set_cellspan(const hpdftbl_t tbl,
                    size_t r, size_t c,
                    size_t rowspan, size_t colspan)
```

The specified  $(r, c)$  is the row and column of the upper left cell in merged cell that spans `rowspan` rows and `colspan` columns. This is also the row and col coordinates used to accessing the combined cell.

To illustrate this we will create a table with seven rows and five columns. We will merge three cells using cell-spanning as follows:

```
hpdftbl_set_cellspan(tbl, 0, 0, 1, 3);
hpdftbl_set_cellspan(tbl, 2, 2, 3, 3);
hpdftbl_set_cellspan(tbl, 3, 0, 4, 1);
```

For the data we will use the same setup as in [tut\\_ex06.c](#) This will then give the result shown in **Figure 8**.



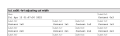
**Figure 8:** \*Having cells spanning multiple rows and columns. [tut\\_ex07.c](#)\*

### 4.2 Adjusting column width

By default, or column widths are divided equally regardless of the content. The width can be adjusted by explicitly set the relative width of a column as a percentage of the total table width. This is done with the function

```
int
hpdftbl_set_colwidth_percent(const hpdftbl_t tbl,
                            const size_t c,
                            const float w);
```

The width is set as a percentage of the total width and is specified as a floating point value in the range  $[0.0, 100.0]$ . An example of this is shown in **Figure 9**. below. An arbitrary number of columns can be given a width. For best result leave at least one column undefined and whatever remains of the table width will be assigned to that column. There is an error to try to specify a total column width  $> 100\%$ .



**Figure 9:** \*Adjusting width of first columns. [tut\\_ex08.c](#) \*





## Chapter 5

# Content and label callbacks

In the "[Getting started](GettingStarted.md)" chapter we discussed the preferred way to specify data and labels in table using data arrays. This is a very good way to populate a table in the cases the data is fairly static.

For data that is more dynamic and determined at runtime it is of course possible to construct the data array but the table library have one better way to do this and that is to set up label and content callbacks.

### 5.1 Introducing content callback functions

**Content callbacks** are functions that are called by the table library for each cell and returns a string which is used as the data to be displayed. The signature for a cell callback is defined by the type `hpdftbl_content_callback_t` which is a pointer to a function defined as:

```
typedef char * (*hpdftbl_content_callback_t)(void *, size_t, size_t);
```

To understand this lets start with a callback function that follows this signature.

```
char *  
my_cell_cb(void *tag, size_t row, size_t col) { ... }
```

The parameters in the callback are

1. **tag**: Since a callback sometimes must know from what table or in what circumstances it is called it is possible to add a "tag" to each table. This could be something as simple as pointer to a numeric identifier that uniquely identifies the table or perhaps a pointer to some function that retrieves data for this particular table. The `tag` for a table is specified with the `hpdftbl_set_tag()` function. When the callback is made this table tag is provided as the first argument.
2. **row**: The cell row
3. **col**: The cell column

It is possible to specify a callback to adjust content, style, and labels. A callback function can be specified either for both the entire table or for individual cells. The API to specify these callbacks are:

1. `hpdftbl_set_content_cb()`:  
Specify a content callback for the entire table.

2. `hpdftbl_set_content_style_cb()`:  
Specify a style callback for the entire table.
3. `hpdftbl_set_label_cb()`:  
Specify a label callback for the entire table.
4. `hpdftbl_set_cell_content_cb()`:  
Specify callback for an individual cell. A cell callback will override a potential table callback.
5. `hpdftbl_set_cell_content_style_cb()`:  
Specify a style callback for an individual cell. A cell callback will override a potential table callback.
6. `hpdftbl_set_canvas_cb()`: This is an advanced callback to allow for low level painting directly on the canvas that is the cell area. The arguments to the callback is different as it includes the bounding-box for the cell area. We will not further discuss this.

#### Note

**Returned content string.** When a content string is added in the table it is added as a copy of the string pointed to by the returned string pointer from the callback function. It is therefore perfectly possible to have a static allocated buffer in the callback function that is used to construct the content. When the table is destroyed using `hpdftbl_destroy()` all used memory will be freed.

## 5.2 A content callback example

Let's now construct a simple example where the content and the labels are specified with callbacks.

We will create callbacks that will add a date string to the top left cell and just some dummy content in the rest of the cells. We could do this in two ways.

1. Add a generic table callback for all cells and then in that callback check if the row and column is (0,0) i.e. top-left and in that case create a date.
2. Add a generic table callback for all cells and then add a specific cell callback with the date for the (0,0) cell.

To illustrate both methods we will use method 1 for the labels and method 2 for the content.

Let's first create the three callback functions we need

```
static char * cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    time_t t = time(NULL);
    ctime_r(&t, buf);
    return buf;
}

static char * cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    snprintf(buf, sizeof buf, "Content %02zu x %02zu", r, c);
    return buf;
}

static char * cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) { // Top-left cell
        snprintf(buf, sizeof buf, "Date:");
    } else {
        snprintf(buf, sizeof buf, "Label %zu x %zu:", r, c);
    }
    return buf;
}
```

We note that we ignore the tag argument. Since we only have one table there is no need to use a tag to different from which table a callback comes.

For the table structure we will re-use our previous example and create a 2x2 table, and we get the following table creation code:

```
void
create_table_ex06(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex06: 2x2 table with callbacks";
    hpdf_tbl_t tbl = hpdf_tbl_create_title(num_rows, num_cols, table_title);
    hpdf_tbl_use_labels(tbl, TRUE);
    hpdf_tbl_use_labelgrid(tbl, TRUE);
    hpdf_tbl_set_label_cb(tbl, cb_labels);
    hpdf_tbl_set_content_cb(tbl, cb_content);
    hpdf_tbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(12);
    HPDF_REAL height = 0; // Calculate height automatically

    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

( tut_ex05.c)
```

Running this example gives the result shown in **Figure 7**. below



Year	Month	Day	Hour
2022	05	08	10

**Figure 7:** Using callbacks to populate the table and labels.



## Chapter 6

# Error handling

All library function will return an error code  $< 0$  and also set a global variable to a specific error code that can later be read by an error handler. In order to translate the error to a human-readable string the function `hpdftbl_get_last_errcode()` can be used as the following error handling snippet exemplified by a call to

```
hpdftbl_set_colwidth_percent()
if( hpdftbl_set_colwidth_percent(tbl, 5, 110) ) {
    // This is an error
    char *err_str;
    int err_code, r, c;
    err_code=hpdftbl_get_last_errcode(&err_str, &r, &c);
    if( err_code ) {
        printf("ERROR*: \"%s\" at cell (%d, %d)",err_str,r,c);
        exit(1);
    }
}
```

As can be seen from the snippet above it would yield quite long winding error handling if one where to check every single library call. Instead, there is the option of installing an error handler that would be called in the event of an error.

The table error handle has the signature

```
void hpdftbl_error_handler_t)(hpdftbl_t tbl, int r, int c, int err)
```

Where the arguments are

1. `tbl` The table in where the error happened. **Note** This might be `NULL` since not all errors happen within the context of a table
2. `r,c` The row and column if the error happens in a specified cell, otherwise these will be `(-1,-1)`
3. `err` The internal error code. This is always a negative number.

The error handler is set with the `hpdftbl_set_errhandler()` method. An example of a very simple error handle is:

```
void
my_table_error_handler(hpdftbl_t t, int r, int c, int err) {
    if( r>-1 && c>-1 ) {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)\n", err, hpdftbl_get_errstr(err), r,
            c);
    } else {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" \n", err, hpdftbl_get_errstr(err));
    }
    exit(1);
}
```

In the above error handler we have made use of the utility function `hpdftbl_get_errstr()` that translates the internal error code to a human-readable string.

In fact this exact error handler is available as a convenience in the library under the name `hpdftbl_default_table_error_handler` so to use this trivial error handler just add the following line to your code

```
hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
```

More advanced error handler must be written for the particular application they are to be used in.

**Note**

A common way to extend the error handling is to log the errors to syslog. When the library is used on OSX from 11.0 and onwards it should be remembered that OSX is broken by design as far as syslog logging is concerned. Apple in its wisdom introduced "Unified logging" which breaks the `syslog()` function and no logging is ever produced in the filesystem directly (i.e. to `/var/log/system.log`).

Instead, the only way to view the logs is by using the utility `log`. So in order to view the log from a particular application the following command has to be given

```
'log stream --info --debug --predicate 'sender == "APPLICATION_NAME"' --style syslog`
```

## 6.1 Translating HPDF error codes

The standard error handler for the plain HPDF library is specified when a new document is created, for example as'

```
...
pdf_doc = HPDF_New(error_handler, NULL);
HPDF_SetCompressionMode(pdf_doc, HPDF_COMP_ALL);
...
```

The error handler signature is defined by Haru PDF library as

```
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data);
```

It is then up to the application code to decide how to handle the error. To simplify the handling of core HPDF error the library also offer a convenience function to translate the Haru library error code into a human-readable string. This function is

```
const char *
hpdftbl_hpdf_get_errstr(const HPDF_STATUS err_code)
```

and is used in the error handler in all the examples.

## 6.2 Example of setting up error handler

The following table creation code have a deliberate error in that it tries to assign a total column width of more than 100% which of course isn't possible.

```
void
create_table_ex10(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    hpdftbl_set_colwidth_percent(tbl, 1, 70);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0; // Calculate height automatically

    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

This is available in the example directory as [tut\\_ex10.c](#). When this code is executed the following will be printed to standard out and the process will be stopped.

```
*** Table Error: [-12] "Total column width exceeds 100%"
```

## Chapter 7

# Style and font setting

The format of each cell can be adjusted with respect to:

1. Font-family and style (size, bold, italic etc.)
2. Font- and background-color
3. Border thickness and color

In this section we will focus on how to adjust the font and background color. The style can be adjusted both for the entire table at once and also for individual cells. The individual cell style will always override the table cell style.

The primary API to adjust the table style are:

```
// Set background color for entire table
int hpdftbl_set_background(hpdftbl_t t,
                           HPDF_RGBColor background);

// Set label style for the entire table
int hpdftbl_set_label_style(hpdftbl_t t,
                            char *font, HPDF_REAL fsize,
                            HPDF_RGBColor color, HPDF_RGBColor background);

// Set content style for entire table
int hpdftbl_set_content_style(hpdftbl_t t,
                              char *font, HPDF_REAL fsize,
                              HPDF_RGBColor color, HPDF_RGBColor background);

// Set content style for specified cell
int hpdftbl_set_cell_content_style(hpdftbl_t t,
                                   size_t r, size_t c,
                                   char *font, HPDF_REAL fsize,
                                   HPDF_RGBColor color, HPDF_RGBColor background);

// Set content style for specified row in table
int hpdftbl_set_row_content_style(hpdftbl_t t,
                                   size_t r,
                                   char *font, HPDF_REAL fsize,
                                   HPDF_RGBColor color, HPDF_RGBColor background);

// Set content style for specified column in table
int hpdftbl_set_col_content_style(hpdftbl_t t,
                                   size_t c,
                                   char *font, HPDF_REAL fsize,
                                   HPDF_RGBColor color, HPDF_RGBColor background);
```

## 7.1 Adjusting fonts and colors

Fonts are specified as a string with the type font family name as recognized by the core Haru PDF library, e.g. "Times-Roman", "Times-Italic", "Times-Bold" etc. As a convenience not to have to remember the exact font name strings the following three font family are defined as `HPDF_FF_*` where the last part of the name is specified as the following table shows

Font family	Italic	Bold	BoldItalic
TIMES	TIMES_ITALIC	TIMES_BOLD	TIMES_BOLDITALIC
HELVETICA	HELVETICA_ITALIC	HELVETICA_BOLD	HELVETICA_BOLDITALIC
COURIER	COURIER_ITALIC	COURIER_BOLD	COURIER_BOLDITALIC

**Table 1:** Predefined font family and variants

So to use the "Helvetica" font family the constant "`HPDF_FF_HELVETICA`" is used and so on.

Colors are specified in the standard Haru way, i.e. as an instance of the structure "`HPDF_RGBColor`". As another convenience the following colors are predefined

```
#define HPDF_COLOR_DARK_RED      (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
#define HPDF_COLOR_RED          (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
#define HPDF_COLOR_LIGHT_GREEN  (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
#define HPDF_COLOR_GREEN        (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
#define HPDF_COLOR_DARK_GRAY    (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
#define HPDF_COLOR_LIGHT_GRAY   (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
#define HPDF_COLOR_GRAY         (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
#define HPDF_COLOR_SILVER       (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
#define HPDF_COLOR_LIGHT_BLUE   (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
#define HPDF_COLOR_BLUE         (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
#define HPDF_COLOR_WHITE        (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
#define HPDF_COLOR_BLACK        (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
```

So for example to set the overall default font to 12pt Times Roman with black text on white bottom the following call must be made

```
...
hpdf_tbl_set_content_style(tbl, HPDF_FF_TIMES, 12, HPDF_COLOR_BLACK, HPDF_COLOR_WHITE);
...
```

Since RGB for colors are specified as a floating point number in range [0.0, 1.0] and most color table give colors as an integer triple there is exists a macro to make this conversion easier

```
#define HPDF_COLOR_FROMRGB(r,g,b) (HPDF_RGBColor){r/255.0,g/255.0,b/255.0}
```

which will allow the easier specification of color such as

```
HPDF_RGBColor color_saddle_brown = HPDF_COLOR_FROMRGB(139,69,19);
```

## 7.2 Using style callbacks

In much the same way as callbacks can be used for specifying content and labels so can a callback be used to specify the style of a cell or the entire table.

A style callback has the following signature

```
_Bool
hpdf_tbl_content_style_callback_t(void *tag, size_t r, size_t c, char *content, hpdf_text_style_t *style);
```

In order for the settings to be applied the callback has to return a boolean TRUE value.

If the callback returns FALSE the settings will **not** be applied.

The parameters are used as follows:

- The `tag` parameter has the same meaning as for content and label callbacks; an optional unique identifier for the table.\*\* The `tag` parameter should always be checked for possible NULL value since it is not required for a table to have a tag.
- The `r` and `c` arguments are the row and column of the cell the callback is made for
- The `content` is the cell content string. The rationale for including this in the style callback is to allow for highlighting in the table of specific data. It could for example be something as simple as wanting to mark all values above a certain threshold with another background color in the table to draw attention.



- Finally, the actual style is encompassed by the `hpdf_text_style_t` and is defined as the following structure

```
typedef struct text_style {
    char *font;
    HPDF_REAL fsize;
    HPDF_RGBColor color;
    HPDF_RGBColor background;
    hpdf_tbl_text_align_t halign;
} hpdf_text_style_t;
```

The style callbacks can exactly as the content callback be specified for either the entire table or for a specific cell. A cell callback will always override a table callback. The two functions to set up style callbacks are

```
int
hpdf_tbl_set_cell_content_style_cb(hpdf_tbl_t tbl,
                                   size_t r, size_t c,
                                   hpdf_tbl_content_style_callback_t cb);

int
hpdf_tbl_set_content_style_cb(hpdf_tbl_t tbl,
                              hpdf_tbl_content_style_callback_t cb);
```

#### Note

Due to some technicalities **the style callbacks are called twice** per cell. The first call is necessary to set up the background canvas and at that stage the content is not necessarily known since it could be later specified with a content callback. The first time the callback is made the `content` parameter is always guaranteed to be `NULL`

### 7.2.1 Style callback example

An example of a callback function to set a background color for a header row/column for a table could for example be done as follows

```
_Bool
cb_style(void *tag, size_t r, size_t c, char *content, hpdf_text_style_t *style) {
    // Format the header row/column with a grey background and Helvetica font while the rest of the
    // table uses "Times Roman"
    if ( 0==r || 0==c ) { // Headers
        style->font = HPDF_FF_HELVETICA_BOLD;
        style->fsize = 12;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_LIGHT_GRAY;
        if ( c > 0 )
            style->halign = CENTER;
        else
            style->halign = LEFT;
    } else { // Content
        style->font = HPDF_FF_TIMES;
        style->fsize = 11;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_WHITE;
        style->halign = CENTER;
    }
    return TRUE;
}
```

and the table setup code can then be written as

```
void
create_table_ex09(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    hpdf_tbl_set_content_cb(tbl, cb_content);
    hpdf_tbl_set_content_style_cb(tbl, cb_style);
    hpdf_tbl_set_colwidth_percent(tbl, 0, 40);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

The resulting table is shown in **Figure 10**. below.

	Header 01	Header 02	Header 03
Extra long Header 04	Content 01	Content 02	Content 03
Extra long Header 05	Content 01	Content 02	Content 03
Extra long Header 06	Content 01	Content 02	Content 03

**Figure 10:** Using a style callback to highlight header rows & columns. [tut\\_ex09.c](#)

## 7.3 Using style themes

A theme (or style theme) is a definition of the "look & feel" of a table. It doesn't affect the structure of the table such as the size of the table or how many columns or rows a cell spans. It is practical shortcut when many tables should be displayed in the same style. It allows the compact specification of the table by applying a theme to the table instead of having to call multiple functions to achieve the same thing. In addition, if the design should be changed there is only one place to update instead of for each table.

### Note

There is not yet any support to read and write themes from a file. A theme is therefor an *in memory* structure useful within one program.

A theme controls the following aspects of a table

- The content and label text style
- The header and title text style
- The inner and outer border style
- The usage (or not) of labels and whether the shorter label grind lines should be used
- If a header row should be used or not
- If a title should be used or not

if you have multiple table in a document it is possible to create a *table theme* which consists of some core styling of a table that can be reused.

All information for a theme is encapsulated in the `hpdftbl_theme` structure.

This structure can be set up manually and then applied to a table. However, the recommended way is to first use the "theme getter" function to get the default theme and then modify this default theme as needed since it allows you to only have to update the parts affected by a change.

The functions to work with a theme are as follows:

```
// Apply the given theme to a table
int
hpdftbl_apply_theme(hpdftbl_t t, hpdftbl_theme_t *theme);
// Get the default theme into a new allocated structure
hpdftbl_theme_t *
hpdftbl_get_default_theme(void);
// Destroy the memory used by a theme
int
hpdftbl_destroy_theme(hpdftbl_theme_t *theme);
```

### Note

It is the responsibility of the user of the library to destroy the theme structure by ensuring that `hpdftbl_destroy_theme()` is called when a theme goes out of scope.

The default font styles for the default theme are shown in table 1.

Style	Font	Size	Color	Background	Alignment
content	HPDF_FF_COURIER	10	Black	White	Left
label	HPDF_FF_TIMES_ITALIC	9	Dark gray	White	Left
header	HPDF_FF_HELVETICA_BOLD	10	Black	Light gray	Center
title	HPDF_FF_HELVETICA_BOLD	11	Black	Light gray	Left

**Table 1:** Default font styles.

Theme parameter	Default value
use_labels	FALSE
use_label_grid_style	FALSE
use_header_row	FALSE

**Table 2:** Default table structure parameters.

Border	Color	Width (pt)
inner_border	Grey	0.7
outer_grid	Dark Grey	1.0

**Table 3:** Default border parameters.

## 7.4 Adjusting grid line styles

There are four distinct set of grid lines as far as the library is concerned.

1. The outer gridlines (or border) around the table, and
2. The inner vertical grid line
3. The inner horizontal grid line
4. The inner top grid line (not the outer border!)

All these types of gridlines are styled in the same way using the functions

```
int
hpdftbl_set_inner_tgrid_style(hpdftbl_t t,
                             HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);

int
hpdftbl_set_inner_vgrid_style(hpdftbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);

int
hpdftbl_set_inner_hgrid_style(hpdftbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);

int
hpdftbl_set_inner_grid_style(hpdftbl_t t,
                             HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle);
```

Each type of gridlines can be adjusted with line width, color and style. The last function in the list, `hpdftbl_set_inner_grid_style()`, is a convenience function that sets both the vertical and horizontal inner lines in one call.

The table below illustrates the various dashed line styles available and their names. See also [hpdftbl\\_dashstyle](#) and grid style functions [hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#), [hpdftbl\\_set\\_inner\\_vgrid\\_style\(\)](#) and [hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#)

Dash Style	Illustration
LINE_SOLID	xxx
LINE_DOT1	"x_x_x_"

Dash Style	Illustration
LINE_DOT2	x__x__x__
LINE_DOT3	"x____x____x____
LINE_DASH1	xx__xx__xx__
LINE_DASH2	xx____xx____xx____
LINE_DASH3	xxxx__xxxx__xxxx__
LINE_DASH4	xxxx____xxxx____xxxx____
LINE_DASHDOT1	xxxxx__xx__xxxxx__xx__xxxxx__xx__
LINE_DASHDOT2	xxxxxxxx__xxx__xxxxxxxx__xxx__xxxxxxxx__xxx__↵ ——

The following example ([tut\\_ex20.c](#)) makes use of these settings as shown below

```
void
create_table_ex20(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 5;
    const size_t num_cols = 4;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    hpdf_tbl_set_inner_vgrid_style(tbl, 0.7, HPDF_COLOR_DARK_GRAY, LINE_SOLID);
    hpdf_tbl_set_inner_hgrid_style(tbl, 0.8, HPDF_COLOR_GRAY, LINE_DOT1);
    hpdf_tbl_set_inner_tgrid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
    hpdf_tbl_set_outer_grid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(10);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

and when run will result in the following table:

Content 0	Content 1	Content 2	Content 3
Content 4	Content 5	Content 6	Content 7
Content 8	Content 9	Content 10	Content 11
Content 12	Content 13	Content 14	Content 15
Content 16	Content 17	Content 18	Content 19

## 7.5 Adding zebra lines in a table

A common way to make it easier to read a table is to make every other row a different color. This is sometimes known as zebra lines (or rows). This can be easily accomplished in the library by using the functions

```
int
hpdf_tbl_set_zebra(hpdf_tbl_t t, _Bool use, int phase);
int
hpdf_tbl_set_zebra_color(hpdf_tbl_t t, HPDF_RGBColor z1, HPDF_RGBColor z2);
```

The first function is used to enable/disable row coloring and the second to set the first and second color. The `phase` parameter determines if color 1 is used first or if color 2 is used on the first row. Setting `phase` to `tom0` will make the first row use color 1 as background.

The default colors are white and light gray. The following example ([tut\\_ex15.c](#)) shows how this can be done:

```
void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    hpdf_tbl_set_zebra(tbl, TRUE, 1);
}
```

```

HPDF_REAL xpos = hpdftbl_cm2dpi(1);
HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
HPDF_REAL width = hpdftbl_cm2dpi(18);
HPDF_REAL height = 0; // Calculate height automatically
// Stroke the table to the page
hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

```

Running this example will give the following result

[tut\\_ex15.c](#)

Content 0	Content 1	Content 2	Content 3	Content 4
Content 5	Content 6	Content 7	Content 8	Content 9
Content 10	Content 11	Content 12	Content 13	Content 14
Content 15	Content 16	Content 17	Content 18	Content 19
Content 20	Content 21	Content 22	Content 23	Content 24
Content 25	Content 26	Content 27	Content 28	Content 29
Content 30	Content 31	Content 32	Content 33	Content 34

We can make a small modification by setting `phase = 1` (instead of the default 0) to start with color2. In addition, we can adjust the inner horizontal gridlines to have the same extra light gray as the zebra line making them "invisible" by modifying the table setup as follows ([tut\\_ex15\\_1.c](#)).

```

void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    //hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_zebra(tbl, TRUE, 1);
    // Normal inner line (same color as default Zebra to make them "invisible"
    hpdftbl_set_inner_hgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY, LINE_SOLID);
    // Top inner line. Comment this line to get a visible top line
    hpdftbl_set_inner_tgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY, LINE_SOLID);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

```

Running this gives the following result:

[tut\\_ex15\\_1.c](#)

Content 0	Content 1	Content 2	Content 3	Content 4
Content 5	Content 6	Content 7	Content 8	Content 9
Content 10	Content 11	Content 12	Content 13	Content 14
Content 15	Content 16	Content 17	Content 18	Content 19
Content 20	Content 21	Content 22	Content 23	Content 24
Content 25	Content 26	Content 27	Content 28	Content 29
Content 30	Content 31	Content 32	Content 33	Content 34



## Chapter 8

# Tables layout from data

So far we have constructed the layout of table by issuing API calls per table to set up, for example, the column widths and what cells should merge with what other cells and so on. Previously we saw that data to be put in the table could be specified by either directly issuing API calls per cell, using a 2D array that we populate with data and then finally use callbacks to generate the data in the cells.

The final and most powerful way of constructing a table is to define the table structure as data. This *structural data* together with a style theme can completely define a table.

This will allow the dynamic construction of tables with only one API call instead of the multiple call required to construct a table the usual way. It can initially seem more complex but for advanced table this is indeed a much simpler and easy to maintain. In fact, this will allow a table to be defined entirely in a database and makes it possible to adjust the table as the data changes without ever updating the code (or recompile).

### 8.1 Defining a table in data

There are two data structures that are used when defining a table. First there is a data structure for the overall table specifics and then in that structure a structure to specify the layout of each cell. In addition, a theme needs to be defined (see section on Themes). It is possible to omit the theme by specifying `NULL` in which case the default theme will be used.

To stroke a table from data the following API call is used

```
int  
hpdftbl_stroke_from_data(HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t tbl_spec, hpdftbl_theme_t  
*theme);
```

In order to populate the table with suitable data callback functions are used (as described in section ??)

The overall table is first defined as an instance of

```
typedef struct hpdftbl_spec {  
    char *title;  
    _Bool use_header;  
    _Bool use_labels;  
    _Bool use_labelgrid;  
    size_t rows;  
    size_t cols;  
    HPDF_REAL xpos;  
    HPDF_REAL ypos;  
    HPDF_REAL width;  
    HPDF_REAL height;  
    hpdftbl_content_callback_t content_cb;  
    hpdftbl_content_callback_t label_cb;  
    hpdftbl_content_style_callback_t style_cb;  
    hpdftbl_callback_t post_cb;  
    hpdftbl_cell_spec_t *cell_spec;  
} hpdftbl_spec_t;
```

Then each cell (referenced above in the `cell_spec` field) is defined as an instance of

```
typedef struct hpdf_tbl_cell_spec {
    size_t row;
    size_t col;
    unsigned rowspan;
    unsigned colspan;
    char *label;
    hpdf_tbl_content_callback_t content_cb;
    hpdf_tbl_content_callback_t label_cb;
    hpdf_tbl_content_style_callback_t style_cb;
    hpdf_tbl_canvas_callback_t canvas_cb;
} hpdf_tbl_cell_spec_t;
```

## 8.2 A first example of defining table as data

To understand how this is done lets start to define a basic 3x3 table with header row (so 4x3 in total) as data. First we create an instance of the table data

```
hpdf_tbl_spec_t tbl_spec = {
    // Title and header flag
    .title=NULL, .use_header=TRUE,
    // Label and labelgrid flags
    .use_labels=FALSE, .use_labelgrid=FALSE,
    // Row and columns
    .rows=4, .cols=3,
    // Position of the table, xpos and ypos
    .xpos=hpdf_tbl_cm2dpi(1), .ypos=hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
    .width=hpdf_tbl_cm2dpi(15), .height=0,
    // Content and label callback
    .content_cb=cb_content, .label_cb=cb_label,
    // Style and table post creation callback
    .style_cb=NULL, .post_cb=NULL,
    // Pointer to optional cell specifications
    .cell_spec=NULL
};
```

### Note

In the table definition we use the C99 feature of specifying the field name when defining data in a structure.

Then the actual API call is trivial to what we have seen before and consists of only one line of code

```
void
create_table_ex13_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    hpdf_tbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
}
```

The result is as expected and shown in **Figure 13** but with much less code!

Header 0001	Header 0001	Header 0002
Content_0000	Content_0001	Content_0002
Content_0000	Content_0001	Content_0002
Content_0000	Content_0001	Content_0002

**Figure 13:** \*Defining a table with a data structure `tut_ex13_1.c`\*

## 8.3 A second example of defining a table as data

In the previous example we kept it simple didn't specify any format or content for a table cell. Let us therefore create a slightly more complex example where we create a form which easily could be used to display data records from a DB.

The nice thing about separating layout and table structure from the data population in the callbacks is that this can almost be seen as a poor man's model-view-controller where the table structure is completely separate from the

A good way to start designing a table is to make a sketch on how it should look. Our goal is to create the table structure as shown in the empty table in **Figure 14** below


**Figure 14:** Sketch of table to be designed

To get this layout we use a basic table with :



1. Five rows and four columns
2. No header and no title
3. We use labels and label grids

To make it easier to see how to construct the table we can overlay the sketch with a grid shown in blue in **Figure 15**. As can be seen this is a basic 5x4 table where a number of cells span multiple columns.



**Figure 15:** Sketch of table to be designed with 5x4 table overlaid

To start we set up the table specification as in the previous example with necessary changes. We will also need to specify cell specifications this time, and we assume those are available in an array of cell structures called `cell_specs`.

Before we specify the table structure we have one design decision to make. For the callbacks we can either use the table callback for all cells and check row and column to get the appropriate data, or we can add individual callbacks for each cell. The first case has the advantage to only need one callback function (but a lot of tests) and the second that each callback will be small and focused to get the data for that individual cell, but we will need potentially one callback for each cell unless there are commonalities between the cells so one callback can serve multiple cells. Remember that we still get the row and column as arguments in the callback so we will always know exactly for which cell the callback was made.

To keep the size of this example we will use the table callback method for content and specify the label directly in the cell specification. With this decision made we get the following definition cell specifications

```
hpdfbtbl_cell_spec_t cell_specs[] = {
    {.row=0, .col=0, .rowspan=1, .colspan=3,
     .label="Name:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=0, .col=3, .rowspan=1, .colspan=1,
     .label="Date:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=1, .col=0, .rowspan=1, .colspan=4,
     .label="Address:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=2, .col=0, .rowspan=1, .colspan=3,
     .label="City:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=2, .col=3, .rowspan=1, .colspan=1,
     .label="Zip:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=3, .col=0, .rowspan=1, .colspan=4,
     .label="E-mail:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=4, .col=0, .rowspan=1, .colspan=2,
     .label="Work-phone:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=4, .col=2, .rowspan=1, .colspan=2,
     .label="Mobile:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    HPDFBTBL_END_CELLSPECS // Sentinel to mark the end of
};
```

As can be seen we need to have an end of cell specification sentinel since we could decide to provide details for one or more cells and there is no way for the library to know how many fields to read otherwise. There is even a convenience constant in the library `PDFBTBL_END_CELLSPECS` that can be used as the last record.

The overall table specification is pretty much as before but with the added cell specifications.

```
hpdfbtbl_spec_t tbl_spec = {
    // Title and header flag
    .title=NULL, .use_header=FALSE,
    // Label and labelgrid flags
    .use_labels=TRUE, .use_labelgrid=TRUE,
    // Row and columns
    .rows=5, .cols=4,
    // xpos and ypos
    .xpos=hpdfbtbl_cm2dpi(1), .ypos=hpdfbtbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
    .width=hpdfbtbl_cm2dpi(15), .height=0,
    // Content and label callback
    .content_cb=cb_content, .label_cb=cb_label,
```

```

// Style and table post creation callback
.style_cb=NULL, .post_cb=NULL,
// Pointer to optional cell specifications
.cell_spec=cell_specs
};

```

When this is run (see [tut\\_ex13\\_2.c](#)) it generates the following image, **Figure 13.2**

123	Downer	Mews	London
123	Downer	Mews	London
123	Downer	Mews	London
123	Downer	Mews	London

**Figure 16:** Specifying a table as data with cell specifications.

What remains is to write the proper table content callback that will populate the table. In a real life scenario his data will most likely come from a database but adding that in our example would bring too far. Instead, we will just use some fake static dummy data to illustrate the principle.

Since we have one callback for all cells we need to test from which cell the call come from. Here is a very important point to make. **The row and column number will be the row and cell columns in the original table before any column or row spans was applied.** In this example it means that for example the "Date" field (upper right) will have row=0 and col=3 and **not** (0, 1) !!.

With this information we can write the following (dummy) table callback

```

static char *
cb_content(void *tag, size_t r, size_t c) {
    static char *cell_content[] =
    {
        "Mark Ericson",
        "12 Sep 2021",
        "123 Downer Mews",
        "London",
        "NW2 HB3",
        "mark.p.ericson@myfinemail.com",
        "+44734 354 184 56",
        "+44771 938 137 11"
    };
    if ( 0==r && 0==c) return cell_content[0];
    else if (0==r && 3==c) return cell_content[1];
    else if (1==r && 0==c) return cell_content[2];
    else if (2==r && 0==c) return cell_content[3];
    else if (2==r && 3==c) return cell_content[4];
    else if (3==r && 0==c) return cell_content[5];
    else if (4==r && 0==c) return cell_content[6];
    else if (4==r && 2==c) return cell_content[7];
    else return NULL;
}

```

and we get the (expected) result as shown in **Figure 17** below.

123	Downer	Mews	London
123	Downer	Mews	London
123	Downer	Mews	London
123	Downer	Mews	London

**Figure 17:** Specifying a table as data with cell specifications and "dummy" data.

The alternative of specifying individual callback for each cell would then require that each cell have a callback provided or perhaps even a mix with both a general table callback and selected cell callbacks.

The priority is such that a cell callback will always override a table callback. In the above example the callback for the name field could as an example be

```

static char *
cb_content_name(void *tag, size_t r, size_t c) {
    static char *cell_content = "Mark Ericson";
    return cell_content;
}

```

## Chapter 9

# Widgets

### 9.1 Overview

A feature in the library is the possibility to add widgets in table cell. A widget is used to visualize data value in a cell instead of a numeric value. For example a percentage value can instead be represented by a horizontal bar.

As of this writing the library supports the following five widgets.

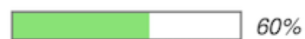
#### 9.1.1 1. Segmented horizontal bar example

Horizontal discrete (segmented) bar. Number of segment is user defined.



#### 9.1.2 2. Horizontal bar example

Basic horizontal bar



#### 9.1.3 3. Signal strength meter example

A widget indicate a signal strength in similar fashion as the signal strength meter on a phone.



### 9.1.4 4. Radio sliding button example

Radio button/Slider with different on/off



### 9.1.5 5. Boxed letters example

Highlight zero or more letters



## 9.2 Widget functions

All the widgets are used in the same way. They are included as a part of a canvas callback function as installed by the `hpdf_tbl_set_canvas_cb()` and `hpdf_tbl_set_cell_canvas_cb()` functions. The callback function itself has to follow the canvas callback signature which is defined as

```
typedef void (*hpdf_tbl_canvas_callback_t)(HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_REAL, HPDF_REAL,
                                           HPDF_REAL,
```

```
                                           HPDF_REAL);
```

and a typical example of a canvas callback function, and its installation would be

```
void
cb_draw_segment_hbar(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                     size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                     HPDF_REAL width, HPDF_REAL height)
{ ... }
...
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_segment_hbar);
```

Each widget has its on function that should be included in the canvas callback to display and size the widget. The different widgets have slightly different defining functions depending on what they display and are defined as follows.

### 9.2.1 Segmented horizontal bar defining function

```
void
hpdf_tbl_widget_segment_hbar(const HPDF_Doc doc, const HPDF_Page page,
                             const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, const
                             HPDF_REAL height,
                             const size_t num_segments, const HPDF_RGBColor on_color, const double
                             val_percent,
                             const _Bool hide_val)
```

### 9.2.2 Horizontal bar defining function

```
void
hpdf_tbl_widget_hbar(const HPDF_Doc doc, const HPDF_Page page,
                     const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, const HPDF_REAL
                     height,
                     const HPDF_RGBColor color, const float val, const _Bool hide_val)
```

### 9.2.3 Signal strength defining function

```
void
hpdftbl_widget_strength_meter(const HPDF_Doc doc, const HPDF_Page page,
                             const HPDF_REAL xpos, const HPDF_REAL ypos, const HPDF_REAL width, const
                             HPDF_REAL height,
                             const size_t num_segments, const HPDF_RGBColor on_color, const size_t
                             num_on_segments)
```

### 9.2.4 Radio sliding button defining function

```
void
hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
                            HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, _Bool state)
```

### 9.2.5 Boxed letters defining function

```
void
hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
                                    HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
                                    const HPDF_RGBColor on_color, const HPDF_RGBColor off_color,
                                    const HPDF_RGBColor on_background, const HPDF_RGBColor off_background,
                                    const HPDF_REAL fsize,
                                    const char *letters, _Bool *state)
```

## 9.3 Usage

The widget function is included in either a table canvas callback or more commonly in a cell canvas callback. Let's construct a basic example with a 1x2 table that shows a segmented horizontal bar indicating a fictive battery charge level and signal strength meter as shown in the figure below



tut_ex14: 2x2 table widget callbacks	
Device name:	Date:
IoT Device ABC123	Wed Apr 27 05:44:29 2022
Battery strength:	Signal:
 40%	

Figure 9.1 tut\_ex14.c

For this we start by constructing the callback for the battery display. In a real application the value would probably be read from a database but here we just use a hard coded value

```
cb_draw_battery_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                      size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                      HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL segment_tot_width = width * 0.5;
    const HPDF_REAL segment_height = height / 3;
    const HPDF_REAL segment_xpos = xpos + 40;
    const HPDF_REAL segment_ypos = ypos + 4;
    const size_t num_segments = 10;
    const _Bool val_text_hide = FALSE; // Display the percentage
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_GREEN;

    // This should in reality be retrieved programmatically (for example from a DB)
    const double val_percent = 0.4;

    hpdftbl_widget_segment_hbar(
        doc, page, segment_xpos, segment_ypos, segment_tot_width,
        segment_height, num_segments, on_color, val_percent, val_text_hide);
}
```

Some comments:

- In the callback we get the bounding box for the cell as arguments

- We adjust the position and height/width so that the widget is centered in the cell

The next callback is the signal strength widget, and we construct that as follows

```
void
cb_draw_signal_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                      size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                      HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL wwidth = 35;
    const HPDF_REAL wheight = 20;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 4;
    const size_t num_segments = 5;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_RED;
    // This should in reality be retrieved programmatically (for example from a DB)
    const size_t num_on_segments = 3;
    hpdftbl_widget_strength_meter(doc, page, wxpos, wypos, wwidth, wheight,
                                  num_segments, on_color, num_on_segments);
}
```

Some comments:

- In the callback we get the bounding box for the cell as arguments
- We adjust the position and height/width so that the widget is centered in the cell

With these callbacks it is now straightforward to construct the table with as follows

```
void
create_table_ex14(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex14: 2x2 table widget callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    // Use one label callback for the entire table
    hpdftbl_set_label_cb(tbl, cb_labels);
    // Name in top left corner
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_device_name);
    // Date in top right corner
    hpdftbl_set_cell_content_cb(tbl, 0, 1, cb_date);
    // Draw battery strength
    hpdftbl_set_cell_canvas_cb(tbl, 1, 0, cb_draw_battery_widget);
    // Draw signal strength
    hpdftbl_set_cell_canvas_cb(tbl, 1, 1, cb_draw_signal_widget);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
```

Some comments:

- For brevity, we have not shown the label and other content callback.
- The complete code is available as [tut\\_ex14.c](#)

## Chapter 10

# HPDFTBL API Overview

### 10.1 Table creation related functions

These calls relate to the creation, destruction and stroking of the table on the PDF page.

- [hpdftbl\\_create\(\)](#) *Create a handle for a new table.*
- [hpdftbl\\_create\\_title\(\)](#) *Create a handle for a new with a title.*
- [hpdftbl\\_destroy\(\)](#) *Destroy (return) memory used by a table.*
- [hpdftbl\\_stroke\(\)](#) *Stroke a table on the specified PDF page.*
- [hpdftbl\\_stroke\\_from\\_data\(\)](#) *Construct and stroke a table defined as a data structure.*
- [hpdftbl\\_get\\_last\\_auto\\_height\(\)](#) *Get the height of the last table stroked.*
- [hpdftbl\\_set\\_anchor\\_top\\_left\(\)](#) *Switch the anchor point of a table between top left and bottom left corner.*
- [hpdftbl\\_get\\_anchor\\_top\\_left\(\)](#) *Get the current anchor point of table.*

### 10.2 Table error handling

- [hpdftbl\\_set\\_errhandler\(\)](#) *Set and error handler callback.*
- [hpdftbl\\_get\\_errstr\(\)](#) *Translate an error code into a human readable string.*
- [hpdftbl\\_get\\_last\\_errcode\(\)](#) *Get the error code from last error raised*
- [hpdftbl\\_default\\_table\\_error\\_handler\(\)](#) *A default error handler callback that print error to stdout and quits the process.*

### 10.3 Theme handling methods

Themes is a technique to easier specify the look and feel to be re-used for multiple tables.

- [hpdftbl\\_apply\\_theme\(\)](#) *Use the specified theme for look & feel of table*
- [hpdftbl\\_get\\_default\\_theme\(\)](#) *Get the default theme. A good way to start and then modify.*
- [hpdftbl\\_destroy\\_theme\(\)](#) *Free all memory structures used by a theme.*

## 10.4 Table layout adjusting functions

Adjusting the structure of the table (apart from number of rows and columns)

- `hpdtbl_set_colwidth_percent()` Set the column width as a percentage of the entire table width.
- `hpdtbl_set_min_rowheight()` Specify the minimum row height in points
- `hpdtbl_set_bottom_vmargin_factor()` Specify the bottom margin for content as a fraction of the specified fontsize
- `hpdtbl_set_cellspan()` Define a cell to span multiple rows and columns.
- `hpdtbl_clear_spanning()` Remove all previous set cell spanning.

## 10.5 Table style modifying functions

These functions are all about look and feel of the table.

- `hpdtbl_use_labels()` Use labels in each cell.
- `hpdtbl_use_labelgrid()` Use shorter left gridlines that only goes down and cover labels
- `hpdtbl_set_background()` Set cell background color.
- `hpdtbl_set_outer_grid_style()` Set style of the table outer grid lines.
- `hpdtbl_set_inner_grid_style()` Set the style of both vertical and horizontal inner grid lines.
- `hpdtbl_set_inner_vgrid_style()` Set the style of table inner vertical grid lines.
- `hpdtbl_set_inner_hgrid_style()` Set the style of table inner horizontal grid lines.
- `hpdtbl_set_header_style()` Set the style for the table header row.
- `hpdtbl_set_header_halign()` Set the horizontal alignment of the header row.
- `hpdtbl_set_title_halign()` Set horizontal alignment for title.
- `hpdtbl_use_header()` Make the top row a header.
- `hpdtbl_set_label_style()` Set style for cell labels.
- `hpdtbl_set_row_content_style()` Set the content style for an entire row.
- `hpdtbl_set_col_content_style()` Set the content style for an entire column.
- `hpdtbl_set_content_style()` Set the content style for the entire table.
- `hpdtbl_set_cell_content_style()` Set the style for specified cell. This overrides any style on the table level.
- `hpdtbl_set_title_style()` Set the style for the table title.



## 10.6 Content handling

Content in a table can be specified in three ways

1. Manually for each cell by calling the `hpdftbl_set_cell()` function
  2. In one go by creating a 1D data array for all cell
  3. Creating a callback which returns the wanted value
- `hpdftbl_set_cell()` Set content text in specified cell.
  - `hpdftbl_set_tag()` Set the table tag. The tag is a `void *` and can be anything. The tag is the first parameter of all callbacks.
  - `hpdftbl_set_title()` Set title text of table.
  - `hpdftbl_set_labels()` Set label texts for the table from 1D-data array.
  - `hpdftbl_set_content()` Set the content text for the entire table from a 1D-data array.

## 10.7 Callback handling

Callbacks can be specified on both table but also on cell level. The simple rule is that if a cell has a callback that is used, otherwise the table callback is used.

- `hpdftbl_set_content_cb()` Set table content callback.
- `hpdftbl_set_cell_content_cb()` Set cell content callback.
- `hpdftbl_set_cell_content_style_cb()` Set the cell style callback.
- `hpdftbl_set_content_style_cb()` Set the table style callback.
- `hpdftbl_set_label_cb()` Set table label callback.
- `hpdftbl_set_cell_label_cb()` Set the cell label callback.
- `hpdftbl_set_canvas_cb()` Set table canvas callback.
- `hpdftbl_set_cell_canvas_cb()` Set the cell canvas callback.

## 10.8 Text encoding

- `hpdftbl_set_text_encoding()` Specify text encodation to use.
- `hpdftbl_encoding_text_out()` Stroke a text with current encoding.

## 10.9 Misc utility function

- `HPDF_RoundedCornerRectangle()` Draw a rectangle with rounded corners.
- `hpdftbl_stroke_grid()` \*Stroke a grid on the PDF page (entire page). This is useful to position the table on a page. The grid is measured in points i.e. postscript natural units.



## Chapter 11

### Todo List

**Global** [HPDFTBL\\_DEFAULT\\_ZEBRA\\_COLOR1](#)

Implement zebra table coloring

**Global** [HPDFTBL\\_DEFAULT\\_ZEBRA\\_COLOR2](#)

Implement zebra table coloring



## Chapter 12

# Data Structure Index

### 12.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">grid_style</a>	Specification for table grid lines . . . . .	51
<a href="#">hpdftbl</a>	Core table handle . . . . .	52
<a href="#">hpdftbl_cell</a>	Specification of individual cells in the table . . . . .	60
<a href="#">hpdftbl_cell_spec</a>	Used in data driven table creation . . . . .	63
<a href="#">hpdftbl_errcode_entry</a>	An entry in the error string table . . . . .	66
<a href="#">hpdftbl_spec</a>	Used in data driven table creation . . . . .	67
<a href="#">hpdftbl_theme</a>	Define a set of styles into a table theme . . . . .	70
<a href="#">line_dash_style</a>	Definition of a dashed line style . . . . .	74
<a href="#">text_style</a>	Specification of a text style . . . . .	75



## Chapter 13

# File Index

### 13.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">bootstrap.sh</a>	Bootstrap the autotools environment and configure a build setup . . . . .	79
<a href="#">dbgblld.sh</a>	Setup a build environment for debugging . . . . .	79
<a href="#">docupload.sh.in</a>	Upload the generated documentation to the github pages doc site for the author . . . . .	80
<a href="#">stdbld.sh</a>	Setup a build environment for production build . . . . .	81
<a href="#">config.h</a>	. . . . .	82
<a href="#">hpdftbl.c</a>	Main module for flexible table drawing with HPDF library . . . . .	83
<a href="#">hpdftbl.h</a>	Header file for libhpdftbl . . . . .	120
<a href="#">hpdftbl_errstr.c</a>	Utility module to translate HPDF error codes to human readable strings . . . . .	175
<a href="#">hpdftbl_grid.c</a>	Create a grid on a document for positioning . . . . .	176
<a href="#">hpdftbl_theme.c</a>	Functions for theme handling . . . . .	177
<a href="#">hpdftbl_widget.c</a>	Support for drawing widgets . . . . .	182





## Chapter 14

# Data Structure Documentation

### 14.1 grid\_style Struct Reference

Specification for table grid lines.

```
#include <hpdftbl.h>
```

#### Data Fields

- HPDF\_REAL [width](#)
- HPDF\_RGBColor [color](#)
- [hpdftbl\\_line\\_dashstyle\\_t](#) [line\\_dashstyle](#)

#### 14.1.1 Detailed Description

Specification for table grid lines.

Contains line properties used when stroking a grid line

#### 14.1.2 Field Documentation

##### 14.1.2.1 color

```
HPDF_RGBColor color
```

Color of grids

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), [hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#), [hpdftbl\\_set\\_inner\\_tgrid\\_style\(\)](#), [hpdftbl\\_set\\_inner\\_vgrid\\_style\(\)](#) and [hpdftbl\\_set\\_outer\\_grid\\_style\(\)](#).

### 14.1.2.2 line\_dashstyle

`hpdf_tbl_line_dashstyle_t line_dashstyle`

Line style for grid

Referenced by [hpdf\\_tbl\\_apply\\_theme\(\)](#), [hpdf\\_tbl\\_set\\_inner\\_hgrid\\_style\(\)](#), [hpdf\\_tbl\\_set\\_inner\\_tgrid\\_style\(\)](#), [hpdf\\_tbl\\_set\\_inner\\_vgrid\\_style\(\)](#) and [hpdf\\_tbl\\_set\\_outer\\_grid\\_style\(\)](#).

### 14.1.2.3 width

`HPDF_REAL width`

Line width of grids

Referenced by [hpdf\\_tbl\\_apply\\_theme\(\)](#), [hpdf\\_tbl\\_set\\_inner\\_hgrid\\_style\(\)](#), [hpdf\\_tbl\\_set\\_inner\\_tgrid\\_style\(\)](#), [hpdf\\_tbl\\_set\\_inner\\_vgrid\\_style\(\)](#) and [hpdf\\_tbl\\_set\\_outer\\_grid\\_style\(\)](#).

The documentation for this struct was generated from the following file:

- [hpdf\\_tbl.h](#)

## 14.2 hpdf\_tbl Struct Reference

Core table handle.

```
#include <hpdf_tbl.h>
```

### Data Fields

- HPDF\_Doc [pdf\\_doc](#)
- HPDF\_Page [pdf\\_page](#)
- size\_t [cols](#)
- size\_t [rows](#)
- HPDF\_REAL [posx](#)
- HPDF\_REAL [posy](#)
- HPDF\_REAL [height](#)
- HPDF\_REAL [minheight](#)
- HPDF\_REAL [bottom\\_vmargin\\_factor](#)
- HPDF\_REAL [width](#)
- void \* [tag](#)
- char \* [title\\_txt](#)
- [hpdf\\_text\\_style\\_t](#) [title\\_style](#)
- [hpdf\\_text\\_style\\_t](#) [header\\_style](#)
- \_Bool [use\\_header\\_row](#)
- [hpdf\\_text\\_style\\_t](#) [label\\_style](#)
- \_Bool [use\\_cell\\_labels](#)
- \_Bool [use\\_label\\_grid\\_style](#)

- [hpdftbl\\_content\\_callback\\_t](#) label\_cb
- [hpdf\\_text\\_style\\_t](#) content\_style
- [hpdftbl\\_content\\_callback\\_t](#) content\_cb
- [hpdftbl\\_content\\_style\\_callback\\_t](#) content\_style\_cb
- [hpdftbl\\_canvas\\_callback\\_t](#) canvas\_cb
- [hpdftbl\\_cell\\_t](#) \* cells
- [hpdftbl\\_grid\\_style\\_t](#) outer\_grid
- [hpdftbl\\_grid\\_style\\_t](#) inner\_vgrid
- [hpdftbl\\_grid\\_style\\_t](#) inner\_hgrid
- [hpdftbl\\_grid\\_style\\_t](#) inner\_tgrid
- [\\_Bool](#) use\_zebra
- [int](#) zebra\_phase
- [HPDF\\_RGBColor](#) zebra\_color1
- [HPDF\\_RGBColor](#) zebra\_color2
- [float](#) \* col\_width\_percent

### 14.2.1 Detailed Description

Core table handle.

This is the main structure that contains all information for the table. The basic structure is an array of cells.

See also

[hpdftbl\\_cell\\_t](#)

Examples

[example01.c](#), [tut\\_ex01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex14.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

### 14.2.2 Field Documentation

#### 14.2.2.1 bottom\_vmargin\_factor

`HPDF_REAL bottom_vmargin_factor`

The content text bottom margin as a factor of the fontsize

Referenced by [hpdftbl\\_set\\_bottom\\_vmargin\\_factor\(\)](#).

#### 14.2.2.2 canvas\_cb

`hpdftbl_canvas_callback_t` canvas\_cb

Table canvas callback. Will be called for each cell unless the cell has its own canvas callback

Referenced by [hpdftbl\\_set\\_canvas\\_cb\(\)](#).

#### 14.2.2.3 cells

`hpdftbl_cell_t*` cells

Reference to all an array of cells in the table

Referenced by [hpdftbl\\_clear\\_spanning\(\)](#), [hpdftbl\\_create\\_title\(\)](#), [hpdftbl\\_set\\_content\(\)](#), and [hpdftbl\\_set\\_labels\(\)](#).

#### 14.2.2.4 col\_width\_percent

`float*` col\_width\_percent

User specified column width array as fraction of the table width. Defaults to equ-width

Referenced by [hpdftbl\\_create\\_title\(\)](#), and [hpdftbl\\_set\\_colwidth\\_percent\(\)](#).

#### 14.2.2.5 cols

`size_t` cols

Number of columns in table

Referenced by [hpdftbl\\_clear\\_spanning\(\)](#), [hpdftbl\\_create\\_title\(\)](#), [hpdftbl\\_destroy\(\)](#), [hpdftbl\\_set\\_colwidth\\_percent\(\)](#), [hpdftbl\\_set\\_content\(\)](#), [hpdftbl\\_set\\_labels\(\)](#), and [hpdftbl\\_set\\_row\\_content\\_style\(\)](#).

#### 14.2.2.6 content\_cb

`hpdftbl_content_callback_t` content\_cb

Table content callback. Will be called for each cell unless the cell has its own content callback

Referenced by [hpdftbl\\_set\\_content\\_cb\(\)](#).

### 14.2.2.7 content\_style

`hpdf_text_style_t` content\_style

Content style

Referenced by [hpdftbl\\_set\\_background\(\)](#), and [hpdftbl\\_set\\_content\\_style\(\)](#).

### 14.2.2.8 content\_style\_cb

`hpdftbl_content_style_callback_t` content\_style\_cb

Style for content callback. Will be called for each cell unless the cell has its own content style callback

Referenced by [hpdftbl\\_set\\_content\\_style\\_cb\(\)](#).

### 14.2.2.9 header\_style

`hpdf_text_style_t` header\_style

Header style

Referenced by [hpdftbl\\_set\\_header\\_halign\(\)](#), and [hpdftbl\\_set\\_header\\_style\(\)](#).

### 14.2.2.10 height

`HPDF_REAL` height

Table height. If specified as 0 then the height will be automatically calculated

### 14.2.2.11 inner\_hgrid

`hpdftbl_grid_style_t` inner\_hgrid

Table inner horizontal border settings, if width>0 this takes precedence over the generic inner border

Referenced by [hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#).

#### 14.2.2.12 inner\_tgrid

`hpdftbl_grid_style_t` inner\_tgrid

Table inner horizontal top border settings, if width>0 this takes precedence over the generic horizontal and inner horizontal border

Referenced by [hpdftbl\\_set\\_inner\\_tgrid\\_style\(\)](#).

#### 14.2.2.13 inner\_vgrid

`hpdftbl_grid_style_t` inner\_vgrid

Table inner vertical border settings, if width>0 this takes precedence over the generic inner border

Referenced by [hpdftbl\\_set\\_inner\\_vgrid\\_style\(\)](#).

#### 14.2.2.14 label\_cb

`hpdftbl_content_callback_t` label\_cb

Table content callback. Will be called for each cell unless the cell has its own content callback

Referenced by [hpdftbl\\_set\\_label\\_cb\(\)](#).

#### 14.2.2.15 label\_style

`hpdf_text_style_t` label\_style

Label style settings

Referenced by [hpdftbl\\_set\\_label\\_style\(\)](#).

#### 14.2.2.16 minheight

`HPDF_REAL` minheight

Minimum table height. If specified as 0 it has no effect

Referenced by [hpdftbl\\_set\\_min\\_rowheight\(\)](#).

### 14.2.2.17 outer\_grid

`hpdftbl_grid_style_t` `outer_grid`

Table outer border settings

Referenced by [hpdftbl\\_set\\_outer\\_grid\\_style\(\)](#).

### 14.2.2.18 pdf\_doc

`HPDF_Doc` `pdf_doc`

PDF document references

### 14.2.2.19 pdf\_page

`HPDF_Page` `pdf_page`

PDF page reference

Referenced by [hpdftbl\\_set\\_line\\_dash\(\)](#).

### 14.2.2.20 posx

`HPDF_REAL` `posx`

X-position of table. Reference point defaults to lower left but can be changed by calling [hpdftbl\\_set\\_anchor\\_top\\_left\(\)](#)

### 14.2.2.21 posy

`HPDF_REAL` `posy`

Y-position of table. Reference point defaults to lower left but can be changed by calling [hpdftbl\\_set\\_anchor\\_top\\_left\(\)](#)

### 14.2.2.22 rows

`size_t` `rows`

Number of rows in table

Referenced by [hpdftbl\\_clear\\_spanning\(\)](#), [hpdftbl\\_create\\_title\(\)](#), [hpdftbl\\_destroy\(\)](#), [hpdftbl\\_set\\_col\\_content\\_style\(\)](#), [hpdftbl\\_set\\_content\(\)](#), and [hpdftbl\\_set\\_labels\(\)](#).

#### 14.2.2.23 tag

`void* tag`

Optional tag used in callbacks. This can be used to identify the table or add any reference needed by a particular application

Referenced by [hpdtbl\\_set\\_tag\(\)](#).

#### 14.2.2.24 title\_style

`hpdtbl_text_style_t title_style`

Title style

Referenced by [hpdtbl\\_set\\_title\\_halign\(\)](#), and [hpdtbl\\_set\\_title\\_style\(\)](#).

#### 14.2.2.25 title\_txt

`char* title_txt`

Title text

Referenced by [hpdtbl\\_create\\_title\(\)](#), [hpdtbl\\_destroy\(\)](#), and [hpdtbl\\_set\\_title\(\)](#).

#### 14.2.2.26 use\_cell\_labels

`_Bool use_cell_labels`

Flag to determine if cell labels should be used

Referenced by [hpdtbl\\_apply\\_theme\(\)](#), and [hpdtbl\\_use\\_labels\(\)](#).

#### 14.2.2.27 use\_header\_row

`_Bool use_header_row`

Flag to determine if the first row in the table should be formatted as a header row

Referenced by [hpdtbl\\_apply\\_theme\(\)](#), and [hpdtbl\\_use\\_header\(\)](#).



#### 14.2.2.28 use\_label\_grid\_style

`_Bool use_label_grid_style`

Flag to determine if the short vertical label border should be used. Default is to use half grid.

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), [hpdftbl\\_use\\_labelgrid\(\)](#), and [hpdftbl\\_use\\_labels\(\)](#).

#### 14.2.2.29 use\_zebra

`_Bool use_zebra`

Use alternating background color on every second line TRUE or FALSE. Defaults to FALSE.

See also

[hpdftbl\\_set\\_zebra\(\)](#)

Referenced by [hpdftbl\\_set\\_zebra\(\)](#).

#### 14.2.2.30 width

`HPDF_REAL width`

Table width

#### 14.2.2.31 zebra\_color1

`HPDF_RGBColor zebra_color1`

First zebra color.

See also

[hpdftbl\\_set\\_zebra\\_color\(\)](#)

Referenced by [hpdftbl\\_set\\_zebra\\_color\(\)](#).

### 14.2.2.32 zebra\_color2

HPDF\_RGBColor zebra\_color2

Second zebra color.

See also

[hpdfctl\\_set\\_zebra\\_color\(\)](#)

Referenced by [hpdfctl\\_set\\_zebra\\_color\(\)](#).

### 14.2.2.33 zebra\_phase

int zebra\_phase

Determine if we start with color1 (phase=0) or start with color2 (phase=1)

See also

[hpdfctl\\_set\\_zebra\(\)](#)

Referenced by [hpdfctl\\_set\\_zebra\(\)](#).

The documentation for this struct was generated from the following file:

- [hpdfctl.h](#)

## 14.3 hpdfctl\_cell Struct Reference

Specification of individual cells in the table.

```
#include <hpdfctl.h>
```

### Data Fields

- char \* [label](#)
- char \* [content](#)
- size\_t [colspan](#)
- size\_t [rowspan](#)
- HPDF\_REAL [height](#)
- HPDF\_REAL [width](#)
- HPDF\_REAL [delta\\_x](#)
- HPDF\_REAL [delta\\_y](#)
- HPDF\_REAL [textwidth](#)
- [hpdfctl\\_content\\_callback\\_t](#) [content\\_cb](#)
- [hpdfctl\\_content\\_callback\\_t](#) [label\\_cb](#)
- [hpdfctl\\_content\\_style\\_callback\\_t](#) [style\\_cb](#)
- [hpdfctl\\_canvas\\_callback\\_t](#) [canvas\\_cb](#)
- [hpdf\\_text\\_style\\_t](#) [content\\_style](#)
- struct [hpdfctl\\_cell](#) \* [parent\\_cell](#)

### 14.3.1 Detailed Description

Specification of individual cells in the table.

This structure contains all information pertaining to each cell in the table. The position of the cell is given as relative position from the lower left corner of the table.

### 14.3.2 Field Documentation

#### 14.3.2.1 canvas\_cb

[hpdfdbl\\_canvas\\_callback\\_t](#) canvas\_cb

Canvas callback. If this is specified then this will override any canvas callback specified for the table

#### 14.3.2.2 colspan

size\_t colspan

Number of column this cell spans

Referenced by [hpdfdbl\\_clear\\_spanning\(\)](#).

#### 14.3.2.3 content

char\* content

String reference for cell content

Referenced by [hpdfdbl\\_set\\_content\(\)](#).

#### 14.3.2.4 content\_cb

[hpdfdbl\\_content\\_callback\\_t](#) content\_cb

Content callback. If this is specified then this will override any content callback specified for the table

#### 14.3.2.5 content\_style

[hpdf\\_text\\_style\\_t](#) content\_style

The style of the text content. If a style callback is specified the callback will override this setting

#### 14.3.2.6 **delta\_x**

HPDF\_REAL delta\_x

X-Position of cell from bottom left of table

#### 14.3.2.7 **delta\_y**

HPDF\_REAL delta\_y

Y-Position of cell from bottom left of table

#### 14.3.2.8 **height**

HPDF\_REAL height

Height of cell

#### 14.3.2.9 **label**

char\* label

String reference for label text

Referenced by [hpdftbl\\_set\\_labels\(\)](#).

#### 14.3.2.10 **label\_cb**

[hpdftbl\\_content\\_callback\\_t](#) label\_cb

Label callback. If this is specified then this will override any content callback specified for the table

#### 14.3.2.11 **parent\_cell**

struct [hpdftbl\\_cell](#)\* parent\_cell

Parent cell. If this cell is part of another cells row or column spanning this is a reference to this parent cell. Normal cells without spanning has NULL as parent cell.

Referenced by [hpdftbl\\_clear\\_spanning\(\)](#).

**14.3.2.12 rowspan**

`size_t rowspan`

Number of rows this cell spans

Referenced by [hpdftbl\\_clear\\_spanning\(\)](#).

**14.3.2.13 style\_cb**

[hpdftbl\\_content\\_style\\_callback\\_t](#) style\_cb

Style for content callback. If this is specified then this will override any style content callback specified for the table

**14.3.2.14 textwidth**

`HPDF_REAL textwidth`

Width of content string

**14.3.2.15 width**

`HPDF_REAL width`

Width of cells

The documentation for this struct was generated from the following file:

- [hpdftbl.h](#)

**14.4 hpdftbl\_cell\_spec Struct Reference**

Used in data driven table creation.

```
#include <hpdftbl.h>
```

**Data Fields**

- `size_t` [row](#)
- `size_t` [col](#)
- unsigned [rowspan](#)
- unsigned [colspan](#)
- `char *` [label](#)
- [hpdftbl\\_content\\_callback\\_t](#) [content\\_cb](#)
- [hpdftbl\\_content\\_callback\\_t](#) [label\\_cb](#)
- [hpdftbl\\_content\\_style\\_callback\\_t](#) [style\\_cb](#)
- [hpdftbl\\_canvas\\_callback\\_t](#) [canvas\\_cb](#)

### 14.4.1 Detailed Description

Used in data driven table creation.

A table can be specified by creating a array of this structure together with the `hpdtbl_spec_t` structure. The array should have one entry for each cell in the table.

See also

[hpdtbl\\_stroke\\_from\\_data\(\)](#)

Examples

[example01.c](#), and [tut\\_ex13\\_2.c](#).

### 14.4.2 Field Documentation

#### 14.4.2.1 canvas\_cb

`hpdtbl_canvas_callback_t` canvas\_cb

Canvas callback for this cell

Referenced by [hpdtbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.2 col

`size_t` col

Row for specified cell

Referenced by [hpdtbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.3 colspan

`unsigned` colspan

Number of columns the specified cell should span

Referenced by [hpdtbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.4 content\_cb

`hpdftbl_content_callback_t` content\_cb

Content callback for this cell

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.5 label

`char*` label

The label for this cell

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.6 label\_cb

`hpdftbl_content_callback_t` label\_cb

Label callback for this cell

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.7 row

`size_t` row

Row for specified cell

##### Examples

[tut\\_ex13\\_2.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.8 rowspan

`unsigned` rowspan

Number of rows the specified cell should span

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.4.2.9 style\_cb

`hpdftbl_content_style_callback_t style_cb`

Content style callback for this cell

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

The documentation for this struct was generated from the following file:

- [hpdftbl.h](#)

### 14.5 hpdftbl\_errcode\_entry Struct Reference

An entry in the error string table.

#### Data Fields

- `char * errstr`
- `unsigned errcode`

#### 14.5.1 Detailed Description

An entry in the error string table.

#### 14.5.2 Field Documentation

##### 14.5.2.1 errcode

`unsigned errcode`

The error code from HPDF library

##### 14.5.2.2 errstr

`char* errstr`

Pointer to the error string

The documentation for this struct was generated from the following file:

- [hpdftbl\\_errstr.c](#)



## 14.6 hpdftbl\_spec Struct Reference

Used in data driven table creation.

```
#include <hpdftbl.h>
```

### Data Fields

- char \* [title](#)
- \_Bool [use\\_header](#)
- \_Bool [use\\_labels](#)
- \_Bool [use\\_labelgrid](#)
- size\_t [rows](#)
- size\_t [cols](#)
- HPDF\_REAL [xpos](#)
- HPDF\_REAL [ypos](#)
- HPDF\_REAL [width](#)
- HPDF\_REAL [height](#)
- [hpdftbl\\_content\\_callback\\_t](#) [content\\_cb](#)
- [hpdftbl\\_content\\_callback\\_t](#) [label\\_cb](#)
- [hpdftbl\\_content\\_style\\_callback\\_t](#) [style\\_cb](#)
- [hpdftbl\\_callback\\_t](#) [post\\_cb](#)
- [hpdftbl\\_cell\\_spec\\_t](#) \* [cell\\_spec](#)

### 14.6.1 Detailed Description

Used in data driven table creation.

This is used together with an array of cell specification [hpdftbl\\_cell\\_spec\\_t](#) to specify the layout of a table.

#### Examples

[example01.c](#), [tut\\_ex13\\_1.c](#), and [tut\\_ex13\\_2.c](#).

### 14.6.2 Field Documentation

#### 14.6.2.1 cell\_spec

```
hpdftbl\_cell\_spec\_t* cell\_spec
```

Array of cell specification

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.2 cols

`size_t cols`

Number of columns in the table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.3 content\_cb

[hpdftbl\\_content\\_callback\\_t](#) `content_cb`

Content callback for this table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.4 height

`HPDF_REAL height`

Height of table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.5 label\_cb

[hpdftbl\\_content\\_callback\\_t](#) `label_cb`

Label callback for this table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.6 post\_cb

[hpdftbl\\_callback\\_t](#) `post_cb`

Post table creation callback. This is an opportunity for a client to do any special table manipulation before the table is stroked to the page. A reference to the table will be passed on in the callback.

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 14.6.2.7 rows

`size_t rows`

Number of rows in the table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 14.6.2.8 style\_cb

`hpdftbl_content_style_callback_t style_cb`

Content style callback for table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 14.6.2.9 title

`char* title`

Table title

#### Examples

[example01.c](#), [tut\\_ex13\\_1.c](#), and [tut\\_ex13\\_2.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 14.6.2.10 use\_header

`_Bool use_header`

Use a header for the table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 14.6.2.11 use\_labelgrid

`_Bool use_labelgrid`

Use label grid in table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.12 use\_labels

`_Bool use_labels`

Use labels in table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.13 width

`HPDF_REAL width`

Width of table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.14 xpos

`HPDF_REAL xpos`

X-position for table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 14.6.2.15 ypos

`HPDF_REAL ypos`

Y-position for table

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

The documentation for this struct was generated from the following file:

- [hpdftbl.h](#)

## 14.7 hpdftbl\_theme Struct Reference

Define a set of styles into a table theme.

```
#include <hpdftbl.h>
```

## Data Fields

- [hpdf\\_text\\_style\\_t](#) `content_style`
- [hpdf\\_text\\_style\\_t](#) `label_style`
- [hpdf\\_text\\_style\\_t](#) `header_style`
- [hpdf\\_text\\_style\\_t](#) `title_style`
- [hpdftbl\\_grid\\_style\\_t](#) `outer_border`
- [\\_Bool](#) `use_labels`
- [\\_Bool](#) `use_label_grid_style`
- [\\_Bool](#) `use_header_row`
- [hpdftbl\\_grid\\_style\\_t](#) `inner_vborder`
- [hpdftbl\\_grid\\_style\\_t](#) `inner_hborder`
- [hpdftbl\\_grid\\_style\\_t](#) `inner_tborder`
- [\\_Bool](#) `use_zebra`
- [int](#) `zebra_phase`
- [HPDF\\_RGBColor](#) `zebra_color1`
- [HPDF\\_RGBColor](#) `zebra_color2`
- [HPDF\\_REAL](#) `bottom_vmargin_factor`

### 14.7.1 Detailed Description

Define a set of styles into a table theme.

Contains all information about the styles of various elements in the table that together make up the table style

Examples

[example01.c](#).

### 14.7.2 Field Documentation

#### 14.7.2.1 `bottom_vmargin_factor`

`HPDF_REAL bottom_vmargin_factor`

Specify the vertical margin factor

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.2 `content_style`

`hpdf_text_style_t content_style`

Content text style

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

### 14.7.2.3 header\_style

`hpdf_text_style_t header_style`

Header text style

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

### 14.7.2.4 inner\_hborder

`hpdftbl_grid_style_t inner_hborder`

Table inner horizontal border settings, if width>0 this takes precedence over the generic inner border

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

### 14.7.2.5 inner\_tborder

`hpdftbl_grid_style_t inner_tborder`

Table inner horizontal top border settings, if width>0 this takes precedence over the generic horizontal and inner horizontal border

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

### 14.7.2.6 inner\_vborder

`hpdftbl_grid_style_t inner_vborder`

Table inner vertical border settings, if width>0 this takes precedence over the generic inner border

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

### 14.7.2.7 label\_style

`hpdf_text_style_t label_style`

Label text style

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.8 outer\_border

`hpdftbl_grid_style_t` outer\_border

Table outer border style

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.9 title\_style

`hpdf_text_style_t` title\_style

Table title text style

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.10 use\_header\_row

`_Bool` use\_header\_row

Flag if header row should be used

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.11 use\_label\_grid\_style

`_Bool` use\_label\_grid\_style

Flag if the special short vertical grid style for labels should be used

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.12 use\_labels

`_Bool` use\_labels

Flag if cell labels should be used

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.13 use\_zebra

`_Bool use_zebra`

Use alternating background color on every second line TRUE or FALSE. Defaults to FALSE.

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), and [hpdfctl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.14 zebra\_color1

`HPDF_RGBColor zebra_color1`

First zebra color.

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), and [hpdfctl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.15 zebra\_color2

`HPDF_RGBColor zebra_color2`

Second zebra color.

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), and [hpdfctl\\_get\\_default\\_theme\(\)](#).

#### 14.7.2.16 zebra\_phase

`int zebra_phase`

Start with color1 or color2

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), and [hpdfctl\\_get\\_default\\_theme\(\)](#).

The documentation for this struct was generated from the following file:

- [hpdfctl.h](#)

## 14.8 line\_dash\_style Struct Reference

Definition of a dashed line style.



## Data Fields

- HPDF\_UINT16 [dash\\_ptn](#) [8]
- [size\\_t](#) [num](#)

### 14.8.1 Detailed Description

Definition of a dashed line style.

### 14.8.2 Field Documentation

#### 14.8.2.1 dash\_ptn

```
HPDF_UINT16 dash_ptn[8]
```

HPDF dash line definition

#### 14.8.2.2 num

```
size_t num
```

Number of segments in the dashed line

The documentation for this struct was generated from the following file:

- [hpdftbl.c](#)

## 14.9 text\_style Struct Reference

Specification of a text style.

```
#include <hpdftbl.h>
```

## Data Fields

- [char \\*](#) [font](#)
- HPDF\_REAL [fsize](#)
- HPDF\_RGBColor [color](#)
- HPDF\_RGBColor [background](#)
- [hpdftbl\\_text\\_align\\_t](#) [halign](#)

### 14.9.1 Detailed Description

Specification of a text style.

This structure collects the basic properties for a text string (font, color, background, horizontal alignment)

Examples

[tut\\_ex09.c](#).

### 14.9.2 Field Documentation

#### 14.9.2.1 background

HPDF\_RGBColor background

Font background color

Examples

[tut\\_ex09.c](#).

Referenced by [hpdtbl\\_apply\\_theme\(\)](#), [hpdtbl\\_set\\_background\(\)](#), [hpdtbl\\_set\\_content\\_style\(\)](#), [hpdtbl\\_set\\_header\\_style\(\)](#), [hpdtbl\\_set\\_label\\_style\(\)](#), and [hpdtbl\\_set\\_title\\_style\(\)](#).

#### 14.9.2.2 color

HPDF\_RGBColor color

Font color

Examples

[tut\\_ex09.c](#).

Referenced by [hpdtbl\\_apply\\_theme\(\)](#), [hpdtbl\\_set\\_content\\_style\(\)](#), [hpdtbl\\_set\\_header\\_style\(\)](#), [hpdtbl\\_set\\_label\\_style\(\)](#), and [hpdtbl\\_set\\_title\\_style\(\)](#).

### 14.9.2.3 font

`char* font`

Font face name

#### Examples

[tut\\_ex09.c](#).

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), [hpdfctl\\_set\\_content\\_style\(\)](#), [hpdfctl\\_set\\_header\\_style\(\)](#), [hpdfctl\\_set\\_label\\_style\(\)](#), and [hpdfctl\\_set\\_title\\_style\(\)](#).

### 14.9.2.4 fsize

`HPDF_REAL fsize`

Font size

#### Examples

[tut\\_ex09.c](#).

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), [hpdfctl\\_set\\_content\\_style\(\)](#), [hpdfctl\\_set\\_header\\_style\(\)](#), [hpdfctl\\_set\\_label\\_style\(\)](#), and [hpdfctl\\_set\\_title\\_style\(\)](#).

### 14.9.2.5 halign

`hpdfctl_text_align_t halign`

Text horizontal alignment

#### Examples

[tut\\_ex09.c](#).

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), [hpdfctl\\_set\\_header\\_halign\(\)](#), and [hpdfctl\\_set\\_title\\_halign\(\)](#).

The documentation for this struct was generated from the following file:

- [hpdfctl.h](#)



# Chapter 15

## File Documentation

### 15.1 bootstrap.sh File Reference

Bootstrap the autotools environment and configure a build setup.

#### Variables

- String **ORIG\_DIR** = "\${PWD}"  
*The original directory from where this script is run.*

#### 15.1.1 Detailed Description

Bootstrap the autotools environment and configure a build setup.

#### Note

This must be run when the source have been obtained by cloning the repo and requires a full installation of GNU autotools as a pre-requisite.

#### Usage:

`bootstrap.sh` [-q] [-h]

-c : Clean **all** generated files. This is equivalent with cloning from the repo.

-q : Quiet

-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson [johan162@gmail.com](mailto:johan162@gmail.com)

### 15.2 dbgbld.sh File Reference

Setup a build environment for debugging.

## Variables

- ReadOnly String **ORIG\_DIR** = "\${PWD}"  
*The original directory from where this script is run.*
- Integer **quiet\_flag** = 0

### 15.2.1 Detailed Description

Setup a build environment for debugging.

In order for easy debugging this means that the debug configuration will only build static library in order to be able to include it in the binaries (e.g. the example programs). With dynamic libraries not yet installed the libtools will build wrapper shell scripts which cannot be debugged.

#### Usage:

`dbgbld.sh [-q] [-h]`

-q : Quiet  
-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson [johan162@gmail.com](mailto:johan162@gmail.com)

## 15.3 docupload.sh.in File Reference

Upload the generated documentation to the github pages doc site for the author.

## Variables

- ReadOnly String **GITHUB\_USER** = "johan162"  
*Specifies the user for github.*
- ReadOnly String **PACKAGE\_NAME** = "@PACKAGE\_NAME@"  
*Specifies the package name. Used to construct the PDF name for the manual.*
- ReadOnly String **VERSION** = "@VERSION@"  
*Defines the version number.*
- ReadOnly String **DOCVERSION** = "v\${VERSION}"  
*The variant of the version number used for documentation.*
- ReadOnly String **PDFNAME** = "\${PACKAGE\_NAME}-\${VERSION}.pdf"  
*The full PDF name.*
- ReadOnly String **COMMIT\_MESSAGE** = "Documentation update for \${PACKAGE\_NAME} \${DOCVERSION}"  
*The git commit message for the doc update.*
- ReadOnly String **GITHUB\_PAGES\_URL** = "git@github.com:\${GITHUB\_USER}/\${GITHUB\_USER}.github.io.git"  
*The full URL for the github pages.*
- ReadOnly String **GITHUB\_PAGES\_REPO** = "\${GITHUB\_USER}.github.io"  
*The repo that corresponds to these pages.*
- ReadOnly String **HTMLDIR\_COPY** = "/docs/out/html"  
*The directory of HTML files to copy to the github pages.*
- ReadOnly String **PDFFILE\_COPY** = "/docs/out/latex/refman.pdf"  
*The PDF file to copy to the github pages.*
- ReadOnly String **ORIG\_DIR** = "\${PWD}"  
*The original directory from where this script is run.*
- Integer **quiet\_flag** = 0

### 15.3.1 Detailed Description

Upload the generated documentation to the github pages doc site for the author.

#### Note

This file is used to generate the actual runnable script via autoconf (e.g. AC\_OUTPUT) as part of the configuration.

#### Usage:

docupload.sh [-q] [-h]

-q : Quiet

-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson [johan162@gmail.com](mailto:johan162@gmail.com)

### 15.3.2 Variable Documentation

#### 15.3.2.1 GITHUB\_USER

```
ReadOnly String GITHUB_USER = "johan162"
```

Specifies the user for github.

This user name dictates the path to the repo as well as the github pages site. The default value here `johan162` corresponds to the authors github account and as such this script will not work without modification for anyone else since the github repos do not have world write permissions.

#### 15.3.2.2 PDFFILE\_COPY

```
ReadOnly String PDFFILE_COPY = "/docs/out/latex/refman.pdf"
```

The PDF file to copy to the github pages.

Note that the name is fixed by Doxygen to `refman.pdf` and is renamed to `PDFNAME` in the copying process.

## 15.4 stdbld.sh File Reference

Setup a build environment for production build.

### Variables

- ReadOnly String **ORIG\_DIR** = "\${PWD}"  
*The original directory from where this script is run.*
- Integer **quiet\_flag** = 0

### 15.4.1 Detailed Description

Setup a build environment for production build.

#### Usage:

`stdbld.sh [-q] [-h]`

`-q` : Quiet

`-h` : Print help and exit

See LICENSE file. (C) 2022 Johan Persson [johan162@gmail.com](mailto:johan162@gmail.com)

## 15.5 config.h

```
1 /* src/config.h. Generated from config.h.in by configure. */
2 /* src/config.h.in. Generated from configure.ac by autoheader. */
3
4 /* Define to 1 if you have the <dlfcn.h> header file. */
5 #define HAVE_DLFCN_H 1
6
7 /* Define to 1 if you have the <hpdf.h> header file. */
8 #define HAVE_HPDPF_H 1
9
10 /* Define to 1 if you have the <iconv.h> header file. */
11 #define HAVE_ICONV_H 1
12
13 /* Define to 1 if you have the <inttypes.h> header file. */
14 #define HAVE_INTTYPES_H 1
15
16 /* Define to 1 if you have the 'hpdf' library (-lhpdf). */
17 #define HAVE_LIBHPDPF 1
18
19 /* Define to 1 if you have the 'iconv' library (-liconv). */
20 #define HAVE_LIBICONV 1
21
22 /* Define to 1 if you have the <stdint.h> header file. */
23 #define HAVE_STDINT_H 1
24
25 /* Define to 1 if you have the <stdio.h> header file. */
26 #define HAVE_STDIO_H 1
27
28 /* Define to 1 if you have the <stdlib.h> header file. */
29 #define HAVE_STDLIB_H 1
30
31 /* Define to 1 if you have the <strings.h> header file. */
32 #define HAVE_STRINGS_H 1
33
34 /* Define to 1 if you have the <string.h> header file. */
35 #define HAVE_STRING_H 1
36
37 /* Define to 1 if you have the <sys/stat.h> header file. */
38 #define HAVE_SYS_STAT_H 1
39
40 /* Define to 1 if you have the <sys/types.h> header file. */
41 #define HAVE_SYS_TYPES_H 1
42
43 /* Define to 1 if you have the <unistd.h> header file. */
44 #define HAVE_UNISTD_H 1
45
46 /* True if system type is Apple OSX */
47 #define IS_OSX 1
48
49 /* Define to the sub-directory where libtool stores uninstalled libraries. */
50 #define LT_OBJDIR ".libs/"
51
52 /* Name of package */
53 #define PACKAGE "libhpdftbl"
54
55 /* Define to the address where bug reports for this package should be sent. */
56 #define PACKAGE_BUGREPORT "johan162@gmail.com"
57
58 /* Define to the full name of this package. */
59 #define PACKAGE_NAME "libhpdftbl"
60
61 /* Define to the full name and version of this package. */
62 #define PACKAGE_STRING "libhpdftbl 1.1.1-wip"
```



```
63
64 /* Define to the one symbol short name of this package. */
65 #define PACKAGE_TARNAME "libhpdfctl"
66
67 /* Define to the home page for this package. */
68 #define PACKAGE_URL ""
69
70 /* Define to the version of this package. */
71 #define PACKAGE_VERSION "1.1.1-wip"
72
73 /* Define to 1 if all of the C90 standard headers exist (not just the ones
74    required in a freestanding environment). This macro is provided for
75    backward compatibility; new code need not use it. */
76 #define STDC_HEADERS 1
77
78 /* Version number of package */
79 #define VERSION "1.1.1-wip"
```

## 15.6 hpdfctl.c File Reference

Main module for flexible table drawing with HPDF library.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <iconv.h>
#include <hpdf.h>
#include "hpdfctl.h"
```

### Data Structures

- struct [line\\_dash\\_style](#)  
*Definition of a dashed line style.*

### Macros

- #define **ERR\_UNKNOWN** 11  
*Error code for unknown error.*

### Typedefs

- typedef struct [line\\_dash\\_style](#) **line\_dash\_style\_t**  
*Definition of a dashed line style.*

## Functions

- `int hpdftbl_set_line_dash (hpdftbl_t t, hpdftbl_line_dashstyle_t style)`  
*Internal helper to set the line style.*
- `void hpdftbl_set_anchor_top_left (const _Bool anchor)`  
*Switch stroking anchor point.*
- `_Bool hpdftbl_get_anchor_top_left (void)`  
*Get stroking anchor point.*
- `const char * hpdftbl_get_errstr (int err)`  
*Translate a table error code to a human readable string.*
- `void hpdftbl_default_table_error_handler (hpdftbl_t t, int r, int c, int err)`  
*A basic default table error handler.*
- `int hpdftbl_get_last_errcode (const char **errstr, int *row, int *col)`  
*Return last error code.*
- `hpdftbl_error_handler_t hpdftbl_set_errhandler (hpdftbl_error_handler_t err_handler)`  
*Specify errhandler for the table routines.*
- `void hpdftbl_set_text_encoding (char *target, char *source)`  
*Determine text source encoding.*
- `int hpdftbl_encoding_text_out (HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, char *text)`  
*Strike text with current encoding.*
- `void HPDF_RoundedCornerRectangle (HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, HPDF_REAL rad)`  
*Draw rectangle with rounded corner.*
- `void hpdftbl_set_bottom_vmargin_factor (hpdftbl_t t, HPDF_REAL f)`  
*The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:*
- `hpdftbl_t hpdftbl_create (size_t rows, size_t cols)`  
*Create a new table with no title.*
- `hpdftbl_t hpdftbl_create_title (size_t rows, size_t cols, char *title)`  
*Create a new table with title top row.*
- `int hpdftbl_set_min_rowheight (hpdftbl_t t, float h)`  
*Set the minimum row height in the table.*
- `int hpdftbl_set_colwidth_percent (hpdftbl_t t, size_t c, float w)`  
*Set column width as percentage of overall table width.*
- `int hpdftbl_set_outer_grid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)`  
*Set outer border grid style.*
- `int hpdftbl_set_inner_grid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)`  
*Set inner border grid style.*
- `int hpdftbl_set_inner_hgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)`  
*Set inner horizontal border grid style.*
- `int hpdftbl_set_inner_vgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)`  
*Set inner vertical border grid style.*
- `int hpdftbl_set_inner_tgrid_style (hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t dashstyle)`  
*Set inner horizontal top border grid style.*
- `int hpdftbl_set_zebra (hpdftbl_t t, _Bool use, int phase)`
- `int hpdftbl_set_zebra_color (hpdftbl_t t, HPDF_RGBColor z1, HPDF_RGBColor z2)`  
*Specify first and second color for a zebra grid table.*

- int [hpdftbl\\_set\\_header\\_style](#) (hpdftbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↵  
RGBColor background)  
*Specify style for table header row.*
- int [hpdftbl\\_set\\_background](#) (hpdftbl\_t t, HPDF\_RGBColor background)  
*Set table background color.*
- int [hpdftbl\\_set\\_header\\_halign](#) (hpdftbl\_t t, hpdftbl\_text\_align\_t align)  
*Set table header horizontal text align.*
- int [hpdftbl\\_use\\_header](#) (hpdftbl\_t t, \_Bool use)  
*Enable/disable the interpretation of the top row as a header row.*
- int [hpdftbl\\_use\\_labels](#) (hpdftbl\_t t, \_Bool use)  
*Enable/Disable the use of cell labels.*
- int [hpdftbl\\_use\\_labelgrid](#) (hpdftbl\_t t, \_Bool use)  
*Shorter vertical line to mark labels.*
- int [hpdftbl\\_set\\_tag](#) (hpdftbl\_t t, void \*tag)  
*Set an optional tag for the table.*
- int [hpdftbl\\_destroy](#) (hpdftbl\_t t)  
*Destroy a table and free all memory.*
- int [hpdftbl\\_set\\_cell](#) (hpdftbl\_t t, int r, int c, char \*label, char \*content)  
*Set content for specific cell.*
- int [hpdftbl\\_set\\_cellspan](#) (hpdftbl\_t t, size\_t r, size\_t c, size\_t rowspan, size\_t colspan)  
*Set cell spanning.*
- int [hpdftbl\\_clear\\_spanning](#) (hpdftbl\_t t)  
*Clear all cell spanning.*
- int [hpdftbl\\_set\\_content\\_cb](#) (hpdftbl\_t t, hpdftbl\_content\_callback\_t cb)  
*Set table content callback.*
- int [hpdftbl\\_set\\_cell\\_content\\_cb](#) (hpdftbl\_t t, size\_t r, size\_t c, hpdftbl\_content\_callback\_t cb)  
*Set cell content callback.*
- int [hpdftbl\\_set\\_cell\\_label\\_cb](#) (hpdftbl\_t t, size\_t r, size\_t c, hpdftbl\_content\_callback\_t cb)  
*Set cell label callback.*
- int [hpdftbl\\_set\\_cell\\_canvas\\_cb](#) (hpdftbl\_t t, size\_t r, size\_t c, hpdftbl\_canvas\_callback\_t cb)  
*Set cell canvas callback.*
- int [hpdftbl\\_set\\_label\\_cb](#) (hpdftbl\_t t, hpdftbl\_content\_callback\_t cb)  
*Set table label callback.*
- int [hpdftbl\\_set\\_canvas\\_cb](#) (hpdftbl\_t t, hpdftbl\_canvas\_callback\_t cb)  
*Set cell canvas callback.*
- int [hpdftbl\\_set\\_labels](#) (hpdftbl\_t t, char \*\*labels)  
*Set the text for the cell labels.*
- int [hpdftbl\\_set\\_content](#) (hpdftbl\_t t, char \*\*content)  
*Set the content for the table.*
- int [hpdftbl\\_set\\_label\\_style](#) (hpdftbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↵  
RGBColor background)  
*Set the style for labels in the entire table.*
- int [hpdftbl\\_set\\_content\\_style](#) (hpdftbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↵  
RGBColor background)  
*Set style for text content.*
- int [hpdftbl\\_set\\_row\\_content\\_style](#) (hpdftbl\_t t, size\_t r, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color,  
HPDF\_RGBColor background)  
*Set the style for an entire row of cells.*
- int [hpdftbl\\_set\\_col\\_content\\_style](#) (hpdftbl\_t t, size\_t c, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color,  
HPDF\_RGBColor background)  
*Set the font style for an entire column of cells.*

- int [hpdftbl\\_set\\_cell\\_content\\_style](#) ([hpdftbl\\_t](#) t, size\_t r, size\_t c, char \*font, HPDF\_REAL fsize, HPDF\_↵  
RGBColor color, HPDF\_RGBColor background)  
*Set the font style for content of specified cell.*
- int [hpdftbl\\_set\\_cell\\_content\\_style\\_cb](#) ([hpdftbl\\_t](#) t, size\_t r, size\_t c, [hpdftbl\\_content\\_style\\_callback\\_t](#) cb)  
*Set cell specific callback to specify cell content style.*
- int [hpdftbl\\_set\\_content\\_style\\_cb](#) ([hpdftbl\\_t](#) t, [hpdftbl\\_content\\_style\\_callback\\_t](#) cb)  
*Set callback to specify cell content style.*
- int [hpdftbl\\_set\\_title\\_style](#) ([hpdftbl\\_t](#) t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↵  
RGBColor background)  
*Set the table title style.*
- int [hpdftbl\\_set\\_title](#) ([hpdftbl\\_t](#) t, char \*title)  
*Set table title.*
- int [hpdftbl\\_set\\_title\\_halign](#) ([hpdftbl\\_t](#) t, [hpdftbl\\_text\\_align\\_t](#) align)  
*Set horizontal alignment for table title.*
- int [hpdftbl\\_stroke\\_from\\_data](#) (HPDF\_Doc pdf\_doc, HPDF\_Page pdf\_page, [hpdftbl\\_spec\\_t](#) \*tbl\_spec,  
[hpdftbl\\_theme\\_t](#) \*theme)  
*Construct the table from a array specification.*
- int [hpdftbl\\_get\\_last\\_auto\\_height](#) (HPDF\_REAL \*height)  
*Get the height calculated for the last constructed table.*
- int [hpdftbl\\_stroke](#) (HPDF\_Doc pdf, const HPDF\_Page page, [hpdftbl\\_t](#) t, const HPDF\_REAL xpos, const  
HPDF\_REAL ypos, const HPDF\_REAL width, HPDF\_REAL height)  
*Stroke the table.*

## Variables

- int **hpdftbl\_err\_code** = 0  
*Stores the last generated error code.*
- int **hpdftbl\_err\_row** = -1  
*The row where the last error was generated.*
- int **hpdftbl\_err\_col** = -1  
*The column where the last error was generated.*

### 15.6.1 Detailed Description

Main module for flexible table drawing with HPDF library.

#### Author

Johan Persson ( [johan162@gmail.com](mailto:johan162@gmail.com) )

Copyright (C) 2022 Johan Persson

See also

LICENSE

Released under the MIT License

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.

## 15.6.2 Function Documentation

### 15.6.2.1 HPDF\_RoundedCornerRectangle()

```
void HPDF_RoundedCornerRectangle (
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    HPDF_REAL width,
    HPDF_REAL height,
    HPDF_REAL rad )
```

Draw rectangle with rounded corner.

Draw a rectangle with rounded corner with the current line width, color. The rectangle will not be stroked.

#### Parameters

<i>page</i>	Page handle
<i>xpos</i>	Lower left x-position of rectangle
<i>ypos</i>	Lower left y-position of rectangle
<i>width</i>	Width of rectangle
<i>height</i>	Height of rectangle
<i>rad</i>	Radius of corners

Referenced by [hpdfdbl\\_widget\\_slide\\_button\(\)](#).

### 15.6.2.2 `hpdftbl_clear_spanning()`

```
int hpdftbl_clear_spanning (
    hpdftbl_t t )
```

Clear all cell spanning.

Reset all spanning cells to no spanning

#### Parameters

<i>t</i>	Table handle
----------	--------------

#### Returns

0 on success, -1 on failure

#### See also

[hpdftbl\\_set\\_cellspan\(\)](#)

### 15.6.2.3 `hpdftbl_create()`

```
hpdftbl_t hpdftbl_create (
    size_t rows,
    size_t cols )
```

Create a new table with no title.

Create a new table structure. This is the basic handler needed for most other API functions.

#### Parameters

<i>rows</i>	Number of rows
<i>cols</i>	Number of columns

#### Returns

A handle to a table, NULL in case of OOM

#### Examples

[tut\\_ex01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

### 15.6.2.4 hpdftbl\_create\_title()

```
hpdftbl_t hpdftbl_create_title (
    size_t rows,
    size_t cols,
    char * title )
```

Create a new table with title top row.

Create a new table structure. This is the basic handler needed for most other API functions.

#### Parameters

<i>rows</i>	Number of rows
<i>cols</i>	Number of columns
<i>title</i>	Title of table

#### Returns

A handle to a table, NULL in case of OOM

#### Examples

[example01.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), and [tut\\_ex14.c](#).

Referenced by [hpdftbl\\_create\(\)](#), and [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.6.2.5 hpdftbl\_default\_table\_error\_handler()

```
void hpdftbl_default_table_error_handler (
    hpdftbl_t t,
    int r,
    int c,
    int err )
```

A basic default table error handler.

This error handler is used as a callback that outputs the error to stderr in human readable format and quits the process.

#### Parameters

<i>t</i>	Table where the error happened (can be NULL)
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>err</i>	The error code

See also

[hpdftbl\\_set\\_errhandler\(\)](#)

Examples

[tut\\_ex10.c](#), [tut\\_ex11.c](#), and [tut\\_ex12.c](#).

### 15.6.2.6 hpdftbl\_destroy()

```
int hpdftbl_destroy (
    hpdftbl_t t )
```

Destroy a table and free all memory.

Destroy a table previous created with `table_create()`, It is the calling routines responsibility not to access `t` again.

Parameters

<i>t</i>	Handle to table
----------	-----------------

Returns

0 on success, -1 on failure

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.6.2.7 hpdftbl\_encoding\_text\_out()

```
int hpdftbl_encoding_text_out (
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    char * text )
```

Stroke text with current encoding.

Utility function to stroke text with character encoding. It is the calling routines responsibility to enclose text in a `HPDF_Page_BeginText()` / `HPDF_Page_EndText()`

Parameters

<i>page</i>	Page handle
<i>xpos</i>	X coordinate
<i>ypos</i>	Y coordinate
<i>text</i>	Text to print



**Returns**

-1 on error, 0 on success

**15.6.2.8 hpdftbl\_get\_anchor\_top\_left()**

```
_Bool hpdftbl_get_anchor_top_left (
    void )
```

Get stroking anchor point.

Get anchor point for table positioning. By default the top left is used.

**See also**

[hpdftbl\\_set\\_anchor\\_top\\_left](#)

**Returns**

TRUE if anchor is top left, FALSE otherwise

**15.6.2.9 hpdftbl\_get\_errstr()**

```
const char * hpdftbl_get_errstr (
    int err )
```

Translate a table error code to a human readable string.

The function returns a pointer to a static string that cannot be modified. It will translate both internal table error messages as well as generic HPDF library error codes.

**Parameters**

<i>err</i>	The error code to be translated
------------	---------------------------------

**Returns**

Static pointer to string for valid error code, NULL otherwise

**See also**

[hpdftbl\\_hpdf\\_get\\_errstr\(\)](#)

Referenced by [hpdftbl\\_default\\_table\\_error\\_handler\(\)](#), and [hpdftbl\\_get\\_last\\_errcode\(\)](#).

### 15.6.2.10 `hpdftbl_get_last_auto_height()`

```
int hpdftbl_get_last_auto_height (
    HPDF_REAL * height )
```

Get the height calculated for the last constructed table.

Get the last automatically calculated height when stroking a table. (The height will be automatically calculated if it was specified as 0)

#### Parameters

<i>height</i>	Returned height
---------------	-----------------

#### Returns

-1 on error, 0 if successful

### 15.6.2.11 `hpdftbl_get_last_errcode()`

```
int hpdftbl_get_last_errcode (
    const char ** errstr,
    int * row,
    int * col )
```

Return last error code.

Return last error code. if *errstr* is not NULL a human readable string describing the error will be copied to the string. The error code will be reset after call.

#### Parameters

<i>errstr</i>	A string buffer where the error string is written to
<i>row</i>	The row where the error was found
<i>col</i>	The col where the error was found

#### Returns

The last error code

#### Examples

[example01.c](#).

### 15.6.2.12 hpdfctl\_set\_anchor\_top\_left()

```
void hpdfctl_set_anchor_top_left (
    const _Bool anchor )
```

Switch stroking anchor point.

Set anchor point for table positioning. By default the top left is used as anchor. Calling this function with FALSE can sets the anchor to bottom left instead.

#### Parameters

<i>anchor</i>	Set to TRUE to use top left as anchor, FALSE for bottom left
---------------	--

### 15.6.2.13 hpdfctl\_set\_background()

```
int hpdfctl_set_background (
    hpdfctl_t t,
    HPDF_RGBColor background )
```

Set table background color.

#### Parameters

<i>t</i>	Table handle
<i>background</i>	Background color

#### Returns

0 on success, -1 on failure

### 15.6.2.14 hpdfctl\_set\_bottom\_vmargin\_factor()

```
void hpdfctl_set_bottom_vmargin_factor (
    hpdfctl_t t,
    HPDF_REAL f )
```

The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:

```
bottom_margin = fontsize * f
```

The default margin is specified by the define [DEFAULT\\_AUTO\\_VBOTTOM\\_MARGIN\\_FACTOR](#)

#### Parameters

<i>t</i>	Table handle
<i>f</i>	Bottom margin factor

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

### 15.6.2.15 hpdftbl\_set\_canvas\_cb()

```
int hpdftbl_set_canvas_cb (
    hpdftbl_t t,
    hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set cell canvas callback. This callback gets called for each cell in the table. The purpose is to allow the client to add dynamic content to the specified cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell. To set the canvas callback only for a specific cell use the [hpdftbl\\_set\\_cell\\_canvas\\_cb\(\)](#) function

#### Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function

#### Returns

-1 on failure, 0 otherwise

#### See also

[hpdftbl\\_set\\_cell\\_canvas\\_cb\(\)](#)

### 15.6.2.16 hpdftbl\_set\_cell()

```
int hpdftbl_set_cell (
    hpdftbl_t t,
    int r,
    int c,
    char * label,
    char * content )
```

Set content for specific cell.

Set label and content for a specific cell. If the specified cell is part of another cells spanning an error occurs (returns -1),

#### Parameters

<i>t</i>	Table handle
<i>r</i>	Row
<i>c</i>	Column
<i>label</i>	Label
<i>content</i>	Text content

**Returns**

-1 on error, 0 if successful

**Examples**

[tut\\_ex01.c](#), and [tut\\_ex03.c](#).

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

**15.6.2.17 hpdfctl\_set\_cell\_canvas\_cb()**

```
int hpdfctl_set_cell_canvas_cb (  
    hpdfctl_t t,  
    size_t r,  
    size_t c,  
    hpdfctl_canvas_callback_t cb )
```

Set cell canvas callback.

Set a canvas callback for an individual cell. This will override the table canvas callback. The canvas callback is called with arguments that give the bounding box for the cell. In that way a callback function may draw arbitrary graphic in the cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell.

**Parameters**

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb</i>	Callback function

**Returns**

-1 on failure, 0 otherwise

**See also**

[hpdfctl\\_canvas\\_callback\\_t](#)  
[hpdfctl\\_set\\_canvas\\_cb\(\)](#)

**Examples**

[example01.c](#), and [tut\\_ex14.c](#).

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

### 15.6.2.18 `hpdfdbl_set_cell_content_cb()`

```
int hpdfdbl_set_cell_content_cb (
    hpdfdbl_t t,
    size_t r,
    size_t c,
    hpdfdbl_content_callback_t cb )
```

Set cell content callback.

Set a content callback for an individual cell. This will override the table content callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

#### Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function
<i>r</i>	Cell row
<i>c</i>	Cell column

#### Returns

-1 on failure, 0 otherwise

#### See also

[hpdfdbl\\_set\\_content\\_cb\(\)](#)

#### Examples

[tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), and [tut\\_ex14.c](#).

Referenced by [hpdfdbl\\_stroke\\_from\\_data\(\)](#).

### 15.6.2.19 `hpdfdbl_set_cell_content_style()`

```
int hpdfdbl_set_cell_content_style (
    hpdfdbl_t t,
    size_t r,
    size_t c,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the font style for content of specified cell.

SSet the font style for content of specified cell. This will override the global cell content setting.

## Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

## Returns

0 on success, -1 on failure

## See also

[hpdftbl\\_set\\_content\\_style\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

## Examples

[example01.c](#).

Referenced by [hpdftbl\\_set\\_col\\_content\\_style\(\)](#), and [hpdftbl\\_set\\_row\\_content\\_style\(\)](#).

## 15.6.2.20 hpdftbl\_set\_cell\_content\_style\_cb()

```
int hpdftbl_set_cell_content_style_cb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    hpdftbl_content_style_callback_t cb )
```

Set cell specific callback to specify cell content style.

Set callback to format the style for the specified cell

## Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb</i>	Callback function

## Returns

0 on success, -1 on failure

See also

[hpdftbl\\_set\\_ontent\\_style\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 15.6.2.21 hpdftbl\_set\_cell\_label\_cb()

```
int hpdftbl_set_cell_label_cb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    hpdftbl_content_callback_t cb )
```

Set cell label callback.

Set a label callback for an individual cell. This will override the table label callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function
<i>r</i>	Cell row
<i>c</i>	Cell column

Returns

-1 on failure, 0 otherwise

See also

[hpdftbl\\_set\\_label\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

#### 15.6.2.22 hpdftbl\_set\_cellspan()

```
int hpdftbl_set_cellspan (
    hpdftbl_t t,
    size_t r,
    size_t c,
    size_t rowspan,
    size_t colspan )
```

Set cell spanning.

Set row and column spanning for a cell, an expanded cell is referenced via the position of it's top-left cell



## Parameters

<i>t</i>	Table handle
<i>r</i>	Row
<i>c</i>	Column
<i>rowspan</i>	Row span
<i>colspan</i>	Column span

## Returns

-1 on error, 0 if successful

## See also

[hpdftbl\\_clear\\_spanning\(\)](#)

## Examples

[example01.c](#), [tut\\_ex07.c](#), and [tut\\_ex08.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

## 15.6.2.23 hpdftbl\_set\_col\_content\_style()

```
int hpdftbl_set_col_content_style (
    hpdftbl_t t,
    size_t c,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the font style for an entire column of cells.

Set font options for the specified column of cells. This will override the global cell content setting.

## Parameters

<i>t</i>	Table handle
<i>c</i>	Column to affect
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

## Returns

0 on success, -1 on failure

See also

[hpdftbl\\_set\\_content\\_style\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

#### 15.6.2.24 hpdftbl\_set\_colwidth\_percent()

```
int hpdftbl_set_colwidth_percent (
    hpdftbl_t t,
    size_t c,
    float w )
```

Set column width as percentage of overall table width.

Specify column width as percentage of total column width. Note that this will only take effect if the table has an overall width specified when stroked. Too avoid errors one column should be left unspecified to let the library use whatever space is left for that column.

##### Parameters

<i>t</i>	Table handle
<i>c</i>	Column to set width of first column has index 0
<i>w</i>	Width as percentage in range [0.0, 100.0]

##### Returns

0 on success, -1 on failure

##### Examples

[example01.c](#), [tut\\_ex08.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), and [tut\\_ex12.c](#).

#### 15.6.2.25 hpdftbl\_set\_content()

```
int hpdftbl_set_content (
    hpdftbl_t t,
    char ** content )
```

Set the content for the table.

Set content for all cells. It is the calling functions responsibility that the content array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r \* num\_cols + c) where num\_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with `strdup()` so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N\*M) entries.

Another way to specify the content is to use the callback mechanism. By setting up a content callback function that returns the content for a cell.

## Parameters

<i>t</i>	Table handle
<i>content</i>	A one dimensional string array of content string

## Returns

-1 on error, 0 if successful

## See also

[hpdftbl\\_set\\_content\\_callback\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_callback\(\)](#)

## Examples

[example01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

## 15.6.2.26 hpdftbl\_set\_content\_cb()

```
int hpdftbl_set_content_cb (
    hpdftbl_t t,
    hpdftbl_content_callback_t cb )
```

Set table content callback.

This callback gets called for each cell in the table and the returned string will be used as the content. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the content will be set to the content specified with the direct content setting. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

## Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function

## Returns

-1 for error , 0 otherwise

## See also

[hpdftbl\\_set\\_cell\\_content\\_cb\(\)](#)

## Examples

[tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), and [tut\\_ex09.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.6.2.27 `hpdftbl_set_content_style()`

```
int hpdftbl_set_content_style (
    hpdftbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set style for text content.

Set style options for cell content (font, color, background). This will be applied for all cells in the table. If a style callback have been specified for either the table or a cell that style take precedence.

#### Parameters

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

#### Returns

-1 on error, 0 if successful

#### See also

[hpdftbl\\_set\\_cell\\_content\\_style\(\)](#)  
[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

#### Examples

[example01.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

### 15.6.2.28 `hpdftbl_set_content_style_cb()`

```
int hpdftbl_set_content_style_cb (
    hpdftbl_t t,
    hpdftbl_content_style_callback_t cb )
```

Set callback to specify cell content style.

Set callback to format the style for cells in the table. If a cell has its own content style callback that callback will override the generic table callback.

## Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function

## Returns

0 on success, -1 on failure

## See also

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

## Examples

[tut\\_ex09.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

**15.6.2.29 hpdftbl\_set\_errhandler()**

```
hpdftbl_error_handler_t hpdftbl_set_errhandler (
    hpdftbl_error_handler_t err_handler )
```

Specify errhandler for the table routines.

Note: The library provides a basic default error handler that can be used,

## Parameters

<i>err_handler</i>	
--------------------	--

## Returns

The old error handler or NULL if non exists

## See also

[hpdftbl\\_default\\_table\\_error\\_handler\(\)](#)

## Examples

[tut\\_ex10.c](#), [tut\\_ex11.c](#), and [tut\\_ex12.c](#).

### 15.6.2.30 `hpdftbl_set_header_halign()`

```
int hpdftbl_set_header_halign (
    hpdftbl_t t,
    hpdftbl_text_align_t align )
```

Set table header horizontal text align.

#### Parameters

<i>t</i>	Table handle
<i>align</i>	Alignment

#### Returns

0 on success, -1 on failure

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

### 15.6.2.31 `hpdftbl_set_header_style()`

```
int hpdftbl_set_header_style (
    hpdftbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Specify style for table header row.

Set the font properties and background for the header row which is the top row if enabled. The header row will be automatically enabled after calling this function. The header can be enabled/disabled separately with [hpdftbl\\_use\\_header\(\)](#)

#### Parameters

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Font color
<i>background</i>	Cell background color

#### Returns

0 on success, -1 on failure

See also

[hpdfctl\\_use\\_header\(\)](#)

Referenced by [hpdfctl\\_apply\\_theme\(\)](#).

### 15.6.2.32 hpdfctl\_set\_inner\_grid\_style()

```
int hpdfctl_set_inner_grid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_line_dashstyle_t dashstyle )
```

Set inner border grid style.

This is a shortform to set both the vertical and horizontal gridline style with one call.

#### Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

#### Returns

0 on success, -1 on failure

See also

[hpdfctl\\_set\\_inner\\_hgrid\\_style\(\)](#), [hpdfctl\\_set\\_inner\\_vgrid\\_style\(\)](#), [hpdfctl\\_set\\_outer\\_grid\\_style\(\)](#)

### 15.6.2.33 hpdfctl\_set\_inner\_hgrid\_style()

```
int hpdfctl_set_inner_hgrid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_line_dashstyle_t dashstyle )
```

Set inner horizontal border grid style.

#### Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#), [hpdftbl\\_set\\_inner\\_vgrid\\_style\(\)](#)

**Examples**

[tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#).

**15.6.2.34 hpdftbl\_set\_inner\_tgrid\_style()**

```
int hpdftbl_set_inner_tgrid_style (
    hpdftbl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdftbl_line_dashstyle_t dashstyle )
```

Set inner horizontal top border grid style.

This would be the gridline just below the header row.

**Parameters**

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#)

**Examples**

[tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).



**15.6.2.35 hpdftbl\_set\_inner\_vgrid\_style()**

```
int hpdftbl_set_inner_vgrid_style (
    hpdftbl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdftbl_line_dashstyle_t dashstyle )
```

Set inner vertical border grid style.

**Parameters**

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#), [hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#)

**Examples**

[tut\\_ex20.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#).

**15.6.2.36 hpdftbl\_set\_label\_cb()**

```
int hpdftbl_set_label_cb (
    hpdftbl_t t,
    hpdftbl_content_callback_t cb )
```

Set table label callback.

Set label callback. This callback gets called for each cell in the table and the returned string will be used as the label. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the label will be set to the content specified with the direct label setting. The callback function will receive the Table tag and the row and column

**Parameters**

<i>t</i>	Table handle
<i>cb</i>	Callback function

**Returns**

-1 on failure, 0 otherwise

**See also**

[hpdfdbl\\_content\\_callback\\_t](#)

[hpdfdbl\\_set\\_cell\\_label\\_cb\(\)](#)

**Examples**

[tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), and [tut\\_ex14.c](#).

Referenced by [hpdfdbl\\_stroke\\_from\\_data\(\)](#).

**15.6.2.37 hpdfdbl\_set\_label\_style()**

```
int hpdfdbl_set_label_style (
    hpdfdbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the style for labels in the entire table.

Set font, color and background options for cell labels. If a style callback have been specified for either the table or a cell that style take precedence.

**Parameters**

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

**Returns**

-1 on error, 0 if successful

Referenced by [hpdfdbl\\_apply\\_theme\(\)](#).

**15.6.2.38 hpdfdbl\_set\_labels()**

```
int hpdfdbl_set_labels (
    hpdfdbl_t t,
    char ** labels )
```

Set the text for the cell labels.

Set labels for all the cell. It is the calling functions responsibility that the labels array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r \* num\_cols + c) where num\_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with `strdup()` so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N\*M) entries.

#### Parameters

<i>t</i>	Table handle
<i>labels</i>	A one dimensional string array of labels

#### Returns

-1 on error, 0 if successful

#### See also

[hpdftbl\\_set\\_cell\\_label\\_cb\(\)](#)

[hpdftbl\\_set\\_label\\_cb\(\)](#)

#### Examples

[example01.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), and [tut\\_ex20.c](#).

### 15.6.2.39 hpdftbl\_set\_line\_dash()

```
int hpdftbl_set_line_dash (
    hpdftbl_t t,
    hpdftbl_line_dashstyle_t style )
```

Internal helper to set the line style.

The drawing of a dashed line uses the underlying HPDF function `HPDF_Page_SetDash()`

#### Parameters

<i>t</i>	Table handle
<i>style</i>	

#### Returns

-1 on error, 0 on success

See also

[line\\_dash\\_style](#)

#### 15.6.2.40 `hpdfdbl_set_min_rowheight()`

```
int hpdfdbl_set_min_rowheight (
    hpdfdbl_t t,
    float h )
```

Set the minimum row height in the table.

The row height is normally calculated based on the font size and if labels are displayed or not. However, it is not possible for the table to know the height of specific widgets (for example) without a two-pass table drawing algorithm.

To handle those odd cases when the calculated height is not sufficient a manual minimum height can be specified.

##### Parameters

<i>t</i>	Table handler
<i>h</i>	The minimum height (in points). If specified as 0 the min height will have no effect.

##### Returns

0 on success, -1 on failure

##### Examples

[example01.c](#).

#### 15.6.2.41 `hpdfdbl_set_outer_grid_style()`

```
int hpdfdbl_set_outer_grid_style (
    hpdfdbl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfdbl_line_dashstyle_t dashstyle )
```

Set outer border grid style.

##### Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#)

**Examples**

[tut\\_ex20.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.6.2.42 hpdftbl\_set\_row\_content\_style()**

```
int hpdftbl_set_row_content_style (
    hpdftbl_t t,
    size_t r,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the style for an entire row of cells.

Set font options for the specified row of cells. This will override the global cell content.

**Parameters**

<i>t</i>	Table handle
<i>r</i>	Row to affect
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_content\\_style\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

#### 15.6.2.43 `hpdfdbl_set_tag()`

```
int hpdfdbl_set_tag (
    hpdfdbl_t t,
    void * tag )
```

Set an optional tag for the table.

Set an optional tag in the table. The tag can be a pointer to anything. The tag is passed as the first argument in the various callbacks and can be used to supply table specific information or identify a specific table in the case the same callback is used for multiple tables.

##### Parameters

<i>t</i>	The table handle
<i>tag</i>	The tag (pointer to any object)

##### Returns

0 on success, -1 on failure

#### 15.6.2.44 `hpdfdbl_set_text_encoding()`

```
void hpdfdbl_set_text_encoding (
    char * target,
    char * source )
```

Determine text source encoding.

The default HPDF encoding is a standard PDF encoding. The problem with that is that now almost 100% of all code is written in UTF-8 encoding and trying to print text strings with accented characters will simply not work. For example the default encoding assumes that strings are given in UTF-8 and sets the target to ISO8859-4 which includes northern europe accented characters. The conversion is internally handled by the standard `iconv()` routines.

##### Parameters

<i>target</i>	The target encoding. See HPDF documentation for supported encodings.
<i>source</i>	The source encodings, i.e. what encodings are sth strings in the source specified in.

#### 15.6.2.45 `hpdfdbl_set_title()`

```
int hpdfdbl_set_title (
    hpdfdbl_t t,
    char * title )
```

Set table title.

Set table title. A title will occupy a separate row above the table that is not included in the row count. A table is enabled when the table text is `<> NULL` and disabled when the title text is `== NULL`.

**Parameters**

<i>t</i>	Table handle
<i>title</i>	Title string

**Returns**

0 on success, -1 on failure

**See also**

[hpdfdbl\\_set\\_title\\_style\(\)](#)

[hpdfdbl\\_set\\_title\\_halign\(\)](#)

**15.6.2.46 hpdfdbl\_set\_title\_halign()**

```
int hpdfdbl_set_title_halign (
    hpdfdbl_t t,
    hpdfdbl_text_align_t align )
```

Set horizontal alignment for table title.

**Parameters**

<i>t</i>	Table handle
<i>align</i>	Alignment

**Returns**

0 on success, -1 on failure

**See also**

[hpdfdbl\\_set\\_title\(\)](#)

[hpdfdbl\\_set\\_title\\_style\(\)](#)

**Examples**

[example01.c](#).

Referenced by [hpdfdbl\\_apply\\_theme\(\)](#).



**15.6.2.47 hpdftbl\_set\_title\_style()**

```
int hpdftbl_set_title_style (
    hpdftbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the table title style.

Set font options for title

**Parameters**

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_title\(\)](#)

[hpdftbl\\_set\\_title\\_halign\(\)](#)

**Examples**

[example01.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.6.2.48 hpdftbl\_set\_zebra()**

```
int hpdftbl_set_zebra (
    hpdftbl_t t,
    _Bool use,
    int phase )
```

**Parameters**

<i>t</i>	Table handle
<i>use</i>	TRUE=Use Zebra, FALSE=Don't use zebra
<i>phase</i>	0=Start with color 1, 1=Start with color 1

**Returns**

0 on successes -1 on failure

**Examples**

[tut\\_ex15.c](#), and [tut\\_ex15\\_1.c](#).

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.6.2.49 hpdftbl\_set\_zebra\_color()**

```
int hpdftbl_set_zebra_color (
    hpdftbl_t t,
    HPDF_RGBColor z1,
    HPDF_RGBColor z2 )
```

Specify first and second color for a zebra grid table.

By default the colors start with `z1` color. To have the top row (below any potential header row) instead start with `z2` specify `phase=1` in the [hpdftbl\\_set\\_zebra\(\)](#) function.

**Parameters**

<i>t</i>	Table handle
<i>z1</i>	Color 1
<i>z2</i>	Color 2

**Returns**

0 on successes -1 on failure

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.6.2.50 hpdftbl\_stroke()**

```
int hpdftbl_stroke (
    HPDF_Doc pdf,
    const HPDF_Page page,
    hpdftbl_t t,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    HPDF_REAL height )
```

Stroke the table.

Stroke the table at the specified position and size. The position is by default specified as the upper left corner of the table. Use the `hpdftbl_set_origin_top_left(FALSE)` to use the bottom left of the table as reference point.

## Parameters

<i>pdf</i>	The HPDF document handle
<i>page</i>	The HPDF page handle
<i>t</i>	Table handle
<i>xpos</i>	x position for table, bottom left corner
<i>ypos</i>	y position for table, bottom left corner
<i>width</i>	width of table
<i>height</i>	height of table. If the height is specified as 0 it will be automatically calculated. The calculated height can be retrieved after the table has been stroked by a call to <a href="#">hpdftbl_get_last_auto_height()</a>

## Returns

-1 on error, 0 if successful

## See also

[hpdftbl\\_get\\_last\\_auto\\_height\(\)](#)

[hpdftbl\\_stroke\\_from\\_data\(\)](#)

## Examples

[example01.c](#), [tut\\_ex01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex14.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

## 15.6.2.51 hpdftbl\_stroke\_from\_data()

```
int hpdftbl_stroke_from_data (
    HPDF_Doc pdf_doc,
    HPDF_Page pdf_page,
    hpdftbl_spec_t * tbl_spec,
    hpdftbl_theme_t * theme )
```

Construct the table from a array specification.

Create and stroke a table specified by a data structure. This makes it easier to separate the view of the data from the model which provides the data. The intended use case is that the data structure specifies the core layout of the table together with the labels and callback functions to handle the content in each cell. Using this method to create a table also makes it much more maintainable.

## Parameters

<i>pdf_doc</i>	The PDF overall document
<i>pdf_page</i>	The pageto stroke to
<i>tbl_spec</i>	The table specification
<i>theme</i>	Table theme to be applied

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_stroke\(\)](#)

**Examples**

[example01.c](#), [tut\\_ex13\\_1.c](#), and [tut\\_ex13\\_2.c](#).

**15.6.2.52 hpdftbl\_use\_header()**

```
int hpdftbl_use_header (
    hpdftbl_t t,
    _Bool use )
```

Enable/disable the interpretation of the top row as a header row.

A header row will have a different style and labels will be disabled on this row. In addition the text will be centered vertically and horizontal in the cell.

**Parameters**

<i>t</i>	Table handle
<i>use</i>	TRUE to enable, FALSE to disable

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_header\\_style\(\)](#)

**Examples**

[example01.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), and [tut\\_ex20.c](#).

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

**15.6.2.53 hpdftbl\_use\_labelgrid()**

```
int hpdftbl_use_labelgrid (
    hpdftbl_t t,
    _Bool use )
```

Shorter vertical line to mark labels.

Set the usage of special grid style where the vertical grid only covers the label text and a gap to the next line. Horizontal lines are drawn as usual. The label grid style gives the table a "lighter" look.

## Parameters

<i>t</i>	Table handle
<i>use</i>	TRUE to use label grid, FALSE o disable it

## Returns

0 on success, -1 on failure

## See also

[hpdfctl\\_use\\_labels\(\)](#)

## Examples

[example01.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex14.c](#), and [tut\\_ex20.c](#).

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

### 15.6.2.54 hpdfctl\_use\_labels()

```
int hpdfctl_use_labels (
    hpdfctl_t t,
    _Bool use )
```

Enable/Disable the use of cell labels.

By default a newly created table will not use cell labels. Enabling labels will also by default enable the special label grid style. To adjust the grid style separately us the [hpdfctl\\_use\\_labelgrid\(\)](#) method.

## Parameters

<i>t</i>	Table handle
<i>use</i>	Set to TRUE for cell labels

## Returns

0 on success, -1 on failure

## See also

[hpdfctl\\_use\\_labelgrid\(\)](#)

## Examples

[example01.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex14.c](#), and [tut\\_ex20.c](#).

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

## 15.7 hpdftbl.h File Reference

Header file for libhpdftbl.

### Data Structures

- struct [text\\_style](#)  
*Specification of a text style.*
- struct [grid\\_style](#)  
*Specification for table grid lines.*
- struct [hpdftbl\\_cell](#)  
*Specification of individual cells in the table.*
- struct [hpdftbl](#)  
*Core table handle.*
- struct [hpdftbl\\_cell\\_spec](#)  
*Used in data driven table creation.*
- struct [hpdftbl\\_spec](#)  
*Used in data driven table creation.*
- struct [hpdftbl\\_theme](#)  
*Define a set of styles into a table theme.*

### Macros

- #define **TRUE** 1  
*Boolean truth value.*
- #define **FALSE** 0  
*Boolean false value.*
- #define **max**(a, b) (((a)>(b)) ? (a):(b))
- #define **min**(a, b) (((a)<(b)) ? (a):(b))
- #define **HPDF\_FF\_TIMES** "Times-Roman"
- #define **HPDF\_FF\_TIMES\_ITALIC** "Times-Italic"
- #define **HPDF\_FF\_TIMES\_BOLD** "Times-Bold"
- #define **HPDF\_FF\_TIMES\_BOLDITALIC** "Times-BoldItalic"
- #define **HPDF\_FF\_HELVETICA** "Helvetica"
- #define **HPDF\_FF\_HELVETICA\_ITALIC** "Helvetica-Oblique"
- #define **HPDF\_FF\_HELVETICA\_BOLD** "Helvetica-Bold"
- #define **HPDF\_FF\_HELVETICA\_BOLDITALIC** "Helvetica-BoldOblique"
- #define **HPDF\_FF\_COURIER** "Courier"
- #define **HPDF\_FF\_COURIER\_BOLD** "Courier-Bold"
- #define **HPDF\_FF\_COURIER\_ITALIC** "Courier-Oblique"
- #define **HPDF\_FF\_COURIER\_BOLDITALIC** "Courier-BoldOblique"
- #define **\_TO\_HPDF\_RGB**(r, g, b) (HPDF\_RGBColor) { r / 255.0f, g / 255.0f, b / 255.0f }  
*Utility macro to create a HPDF color constant from integer RGB values.*
- #define **HPDF\_COLOR\_DARK\_RED** (HPDF\_RGBColor) { 0.6f, 0.0f, 0.0f }
- #define **HPDF\_COLOR\_RED** (HPDF\_RGBColor) { 1.0f, 0.0f, 0.0f }
- #define **HPDF\_COLOR\_LIGHT\_GREEN** (HPDF\_RGBColor) { 0.9f, 1.0f, 0.9f }
- #define **HPDF\_COLOR\_GREEN** (HPDF\_RGBColor) { 0.4f, 0.9f, 0.4f }
- #define **HPDF\_COLOR\_DARK\_GREEN** (HPDF\_RGBColor) { 0.05f, 0.37f, 0.02f }
- #define **HPDF\_COLOR\_DARK\_GRAY** (HPDF\_RGBColor) { 0.2f, 0.2f, 0.2f }
- #define **HPDF\_COLOR\_LIGHT\_GRAY** (HPDF\_RGBColor) { 0.9f, 0.9f, 0.9f }

- `#define HPDF_COLOR_XLIGHT_GRAY (HPDF_RGBColor) { 0.95f, 0.95f, 0.95f }`
- `#define HPDF_COLOR_GRAY (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }`
- `#define HPDF_COLOR_SILVER (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }`
- `#define HPDF_COLOR_LIGHT_BLUE (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }`
- `#define HPDF_COLOR_BLUE (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }`
- `#define HPDF_COLOR_DARK_BLUE (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }`
- `#define HPDF_COLOR_WHITE (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }`
- `#define HPDF_COLOR_BLACK (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }`
- `#define HPDF_COLOR_ORANGE_TO_HPDI_RGB(0xF5, 0xD0, 0x98);`
- `#define HPDF_COLOR_ALMOST_BLACK_TO_HPDI_RGB(0x14, 0x14, 0x14);`
- `#define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5`  
*The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size.*
- `#define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"`  
*Default PDF text encodings.*
- `#define HPDFTBL_DEFAULT_SOURCE_ENCODING "UTF-8"`  
*Default input source text encodings.*
- `#define A4PAGE_HEIGHT_CM 29.7`  
*Standard A4 paper height in cm.*
- `#define A4PAGE_WIDTH_CM 21.0`  
*Standard A4 paper width in cm.*
- `#define A3PAGE_HEIGHT_CM 42.0`  
*Standard A3 paper height in cm.*
- `#define A3PAGE_WIDTH_CM 29.7`  
*Standard A3 paper width in cm.*
- `#define LETTERPAGE_HEIGHT_CM 27.9`  
*US Letter Height in cm.*
- `#define LETTERPAGE_WIDTH_CM 21.6`  
*US Letter width in cm.*
- `#define LEGALPAGE_HEIGHT_CM 35.6`  
*US Legal Height in cm.*
- `#define LEGALPAGE_WIDTH_CM 21.6`  
*US Legal Width in cm.*
- `#define HPDFTBL_END_CELLSPECS {0, 0, 0, 0, 0, 0, 0, 0, 0}`  
*Sentinel to mark the end of Cell Specifications for data driven table definition.*
- `#define HPDF_COLOR_FROMRGB(r, g, b) (HPDF_RGBColor){(r)/255.0,(g)/255.0,(b)/255.0}`  
*Utility macro to calculate a color constant from RGB integer values [0,255].*
- `#define HPDFTBL_MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0`  
*The smallest size in percent of table width allowed by automatic calculation before giving an error.*
- `#define hpdftbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54*72)`  
*Convert cm to dots using the default resolution (72 DPI)*
- `#define _HPDFTBL_SET_ERR(t, err, r, c) do {hpdftbl_err_code=err;hpdftbl_err_row=r;hpdftbl_err_col=c; if(hpdftbl_err_handler){hpdftbl_err_handler(t,r,c,err);} } while(0)`  
*Call the error handler with specified error code and table row, col where error occurred.*
- `#define _HPDFTBL_CHK_TABLE(t) do {if(NULL == t) {hpdftbl_err_code=-3;hpdftbl_err_row=-1;hpdftbl_err_col=-1;return -1;} } while(0)`  
*NPE check before using a table handler.*
- `#define _HPDFTBL_IDX(r, c) (r*t->cols+c)`  
*Shortcut to calculate the index in an array from a row,column (table) position.*

## Typedefs

- typedef enum [hpdf\\_tbl\\_text\\_align](#) [hpdf\\_tbl\\_text\\_align\\_t](#)  
*Enumeration for horizontal text alignment.*
- typedef struct [text\\_style](#) [hpdf\\_text\\_style\\_t](#)  
*Specification of a text style.*
- typedef char \*(\* [hpdf\\_tbl\\_content\\_callback\\_t](#)) (void \*, size\_t, size\_t)  
*Type specification for the table content callback.*
- typedef void(\* [hpdf\\_tbl\\_canvas\\_callback\\_t](#)) (HPDF\_Doc, HPDF\_Page, void \*, size\_t, size\_t, HPDF\_REAL, HPDF\_REAL, HPDF\_REAL, HPDF\_REAL)  
*Type specification for the table canvas callback.*
- typedef \_Bool(\* [hpdf\\_tbl\\_content\\_style\\_callback\\_t](#)) (void \*, size\_t, size\_t, char \*content, [hpdf\\_text\\_style\\_t](#) \*)  
*Type specification for the content style.*
- typedef enum [hpdf\\_tbl\\_dashstyle](#) [hpdf\\_tbl\\_line\\_dashstyle\\_t](#)  
*Possible line dash styles for grid lines.*
- typedef struct [grid\\_style](#) [hpdf\\_tbl\\_grid\\_style\\_t](#)  
*Specification for table grid lines.*
- typedef struct [hpdf\\_tbl\\_cell](#) [hpdf\\_tbl\\_cell\\_t](#)  
*Type definition for the cell structure.*
- typedef struct [hpdf\\_tbl](#) \* [hpdf\\_tbl\\_t](#)  
*Table handle is a pointer to the hpdf\_tbl structure.*
- typedef void(\* [hpdf\\_tbl\\_callback\\_t](#)) ([hpdf\\_tbl\\_t](#))  
*Callback type for optional post processing when constructing table from a data array.*
- typedef struct [hpdf\\_tbl\\_cell\\_spec](#) [hpdf\\_tbl\\_cell\\_spec\\_t](#)  
*Used in data driven table creation.*
- typedef struct [hpdf\\_tbl\\_spec](#) [hpdf\\_tbl\\_spec\\_t](#)  
*Used in data driven table creation.*
- typedef struct [hpdf\\_tbl\\_theme](#) [hpdf\\_tbl\\_theme\\_t](#)  
*Define a set of styles into a table theme.*
- typedef void(\* [hpdf\\_tbl\\_error\\_handler\\_t](#)) ([hpdf\\_tbl\\_t](#), int, int, int)  
*Type for error handler function.*

## Enumerations

- enum [hpdf\\_tbl\\_text\\_align](#) { [LEFT](#) = 0 , [CENTER](#) = 1 , [RIGHT](#) = 2 }
- enum [hpdf\\_tbl\\_dashstyle](#) {  
    [LINE\\_SOLID](#) = 0 , [LINE\\_DOT1](#) = 1 , [LINE\\_DOT2](#) = 2 , [LINE\\_DOT3](#) = 3 ,  
    [LINE\\_DASH1](#) = 4 , [LINE\\_DASH2](#) = 5 , [LINE\\_DASH3](#) = 6 , [LINE\\_DASH4](#) = 7 ,  
    [LINE\\_DASHDOT1](#) = 8 , [LINE\\_DASHDOT2](#) = 9 }
- *Possible line dash styles for grid lines.*

## Functions

- [hpdf\\_tbl\\_t](#) [hpdf\\_tbl\\_create](#) (size\_t rows, size\_t cols)  
*Create a new table with no title.*
- [hpdf\\_tbl\\_t](#) [hpdf\\_tbl\\_create\\_title](#) (size\_t rows, size\_t cols, char \*title)  
*Create a new table with title top row.*
- int [hpdf\\_tbl\\_stroke](#) (HPDF\_Doc pdf, HPDF\_Page page, [hpdf\\_tbl\\_t](#) t, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height)



- Stroke the table.*

  - int [hpdftbl\\_stroke\\_from\\_data](#) (HPDF\_Doc pdf\_doc, HPDF\_Page pdf\_page, [hpdftbl\\_spec\\_t](#) \*tbl\_spec, [hpdftbl\\_theme\\_t](#) \*theme)
- Construct the table from a array specification.*

  - int [hpdftbl\\_destroy](#) ([hpdftbl\\_t](#) t)
- Destroy a table and free all memory.*

  - int [hpdftbl\\_get\\_last\\_auto\\_height](#) (HPDF\_REAL \*height)
- Get the height calculated for the last constructed table.*

  - void [hpdftbl\\_set\\_anchor\\_top\\_left](#) (\_Bool anchor)
- Switch stroking anchor point.*

  - \_Bool [hpdftbl\\_get\\_anchor\\_top\\_left](#) (void)
- Get stroking anchor point.*

  - [hpdftbl\\_error\\_handler\\_t](#) [hpdftbl\\_set\\_errhandler](#) ([hpdftbl\\_error\\_handler\\_t](#))
- Specify errhandler for the table routines.*

  - const char \* [hpdftbl\\_get\\_errstr](#) (int err)
- Translate a table error code to a human readable string.*

  - const char \* [hpdftbl\\_hpdf\\_get\\_errstr](#) (HPDF\_STATUS err\_code)
- Function to return a human readable error string for an error code from Core HPDF library.*

  - int [hpdftbl\\_get\\_last\\_errcode](#) (const char \*\*errstr, int \*row, int \*col)
- Return last error code.*

  - void [hpdftbl\\_default\\_table\\_error\\_handler](#) ([hpdftbl\\_t](#) t, int r, int c, int err)
- A basic default table error handler.*

  - int [hpdftbl\\_apply\\_theme](#) ([hpdftbl\\_t](#) t, [hpdftbl\\_theme\\_t](#) \*theme)
- Apply a specified theme to a table.*

  - [hpdftbl\\_theme\\_t](#) \* [hpdftbl\\_get\\_default\\_theme](#) (void)
- Return the default theme.*

  - int [hpdftbl\\_destroy\\_theme](#) ([hpdftbl\\_theme\\_t](#) \*theme)
- Destroy existing theme structure and free memory.*

  - void [hpdftbl\\_set\\_bottom\\_vmargin\\_factor](#) ([hpdftbl\\_t](#) t, HPDF\_REAL f)
- The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:*

  - int [hpdftbl\\_set\\_min\\_rowheight](#) ([hpdftbl\\_t](#) t, float h)
- Set the minimum row height in the table.*

  - int [hpdftbl\\_set\\_colwidth\\_percent](#) ([hpdftbl\\_t](#) t, size\_t c, float w)
- Set column width as percentage of overall table width.*

  - int [hpdftbl\\_clear\\_spanning](#) ([hpdftbl\\_t](#) t)
- Clear all cell spanning.*

  - int [hpdftbl\\_set\\_cellspan](#) ([hpdftbl\\_t](#) t, size\_t r, size\_t c, size\_t rowspan, size\_t colspan)
- Set cell spanning.*

  - int [hpdftbl\\_set\\_zebra](#) ([hpdftbl\\_t](#) t, \_Bool use, int phase)
- Specify first and second color for a zebra grid table.*

  - int [hpdftbl\\_set\\_zebra\\_color](#) ([hpdftbl\\_t](#) t, HPDF\_RGBColor z1, HPDF\_RGBColor z2)
- Enable/Disable the use of cell labels.*

  - int [hpdftbl\\_use\\_labels](#) ([hpdftbl\\_t](#) t, \_Bool use)
- Enable/Disable the use of cell labels.*

  - int [hpdftbl\\_use\\_labelgrid](#) ([hpdftbl\\_t](#) t, \_Bool use)
- Shorter vertical line to mark labels.*

  - int [hpdftbl\\_set\\_background](#) ([hpdftbl\\_t](#) t, HPDF\_RGBColor background)
- Set table background color.*

  - int [hpdftbl\\_set\\_inner\\_tgrid\\_style](#) ([hpdftbl\\_t](#) t, HPDF\_REAL width, HPDF\_RGBColor color, [hpdftbl\\_line\\_dashstyle\\_t](#) dashstyle)
- Set inner horizontal top border grid style.*

- int [hpdtbl\\_set\\_inner\\_vgrid\\_style](#) (hpdtbl\_t t, HPDF\_REAL width, HPDF\_RGBColor color, [hpdtbl\\_line\\_dashstyle\\_t](#) dashstyle)  
*Set inner vertical border grid style.*
- int [hpdtbl\\_set\\_inner\\_hgrid\\_style](#) (hpdtbl\_t t, HPDF\_REAL width, HPDF\_RGBColor color, [hpdtbl\\_line\\_dashstyle\\_t](#) dashstyle)  
*Set inner horizontal border grid style.*
- int [hpdtbl\\_set\\_inner\\_grid\\_style](#) (hpdtbl\_t t, HPDF\_REAL width, HPDF\_RGBColor color, [hpdtbl\\_line\\_dashstyle\\_t](#) dashstyle)  
*Set inner border grid style.*
- int [hpdtbl\\_set\\_outer\\_grid\\_style](#) (hpdtbl\_t t, HPDF\_REAL width, HPDF\_RGBColor color, [hpdtbl\\_line\\_dashstyle\\_t](#) dashstyle)  
*Set outer border grid style.*
- int [hpdtbl\\_set\\_header\\_style](#) (hpdtbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↔ RGBColor background)  
*Specify style for table header row.*
- int [hpdtbl\\_set\\_header\\_halign](#) (hpdtbl\_t t, [hpdtbl\\_text\\_align\\_t](#) align)  
*Set table header horizontal text align.*
- int [hpdtbl\\_use\\_header](#) (hpdtbl\_t t, \_Bool use)  
*Enable/disable the interpretation of the top row as a header row.*
- int [hpdtbl\\_set\\_label\\_style](#) (hpdtbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↔ RGBColor background)  
*Set the style for labels in the entire table.*
- int [hpdtbl\\_set\\_row\\_content\\_style](#) (hpdtbl\_t t, size\_t r, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_RGBColor background)  
*Set the style for an entire row of cells.*
- int [hpdtbl\\_set\\_col\\_content\\_style](#) (hpdtbl\_t t, size\_t c, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_RGBColor background)  
*Set the font style for an entire column of cells.*
- int [hpdtbl\\_set\\_content\\_style](#) (hpdtbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↔ RGBColor background)  
*Set style for text content.*
- int [hpdtbl\\_set\\_cell\\_content\\_style](#) (hpdtbl\_t t, size\_t r, size\_t c, char \*font, HPDF\_REAL fsize, HPDF\_↔ RGBColor color, HPDF\_RGBColor background)  
*Set the font style for content of specified cell.*
- int [hpdtbl\\_set\\_title\\_style](#) (hpdtbl\_t t, char \*font, HPDF\_REAL fsize, HPDF\_RGBColor color, HPDF\_↔ RGBColor background)  
*Set the table title style.*
- int [hpdtbl\\_set\\_cell](#) (hpdtbl\_t t, int r, int c, char \*label, char \*content)  
*Set content for specific cell.*
- int [hpdtbl\\_set\\_tag](#) (hpdtbl\_t t, void \*tag)  
*Set an optional tag for the table.*
- int [hpdtbl\\_set\\_title](#) (hpdtbl\_t t, char \*title)  
*Set table title.*
- int [hpdtbl\\_set\\_title\\_halign](#) (hpdtbl\_t t, [hpdtbl\\_text\\_align\\_t](#) align)  
*Set horizontal alignment for table title.*
- int [hpdtbl\\_set\\_labels](#) (hpdtbl\_t t, char \*\*labels)  
*Set the text for the cell labels.*
- int [hpdtbl\\_set\\_content](#) (hpdtbl\_t t, char \*\*content)  
*Set the content for the table.*
- int [hpdtbl\\_set\\_content\\_cb](#) (hpdtbl\_t t, [hpdtbl\\_content\\_callback\\_t](#) cb)  
*Set table content callback.*
- int [hpdtbl\\_set\\_cell\\_content\\_cb](#) (hpdtbl\_t t, size\_t r, size\_t c, [hpdtbl\\_content\\_callback\\_t](#) cb)

- Set cell content callback.*

  - int [hpdfctl\\_set\\_cell\\_content\\_style\\_cb](#) (hpdfctl\_t t, size\_t r, size\_t c, [hpdfctl\\_content\\_style\\_callback\\_t](#) cb)

*Set cell specific callback to specify cell content style.*
- int [hpdfctl\\_content\\_style\\_cb](#) (hpdfctl\_t t, [hpdfctl\\_content\\_style\\_callback\\_t](#) cb)

*Set callback to specify cell content style.*
- int [hpdfctl\\_set\\_label\\_cb](#) (hpdfctl\_t t, [hpdfctl\\_content\\_callback\\_t](#) cb)

*Set table label callback.*
- int [hpdfctl\\_set\\_cell\\_label\\_cb](#) (hpdfctl\_t t, size\_t r, size\_t c, [hpdfctl\\_content\\_callback\\_t](#) cb)

*Set cell label callback.*
- int [hpdfctl\\_set\\_canvas\\_cb](#) (hpdfctl\_t t, [hpdfctl\\_canvas\\_callback\\_t](#) cb)

*Set cell canvas callback.*
- int [hpdfctl\\_set\\_cell\\_canvas\\_cb](#) (hpdfctl\_t t, size\_t r, size\_t c, [hpdfctl\\_canvas\\_callback\\_t](#) cb)

*Set cell canvas callback.*
- void [hpdfctl\\_set\\_text\\_encoding](#) (char \*target, char \*source)

*Determine text source encoding.*
- int [hpdfctl\\_encoding\\_text\\_out](#) (HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, char \*text)

*Strike text with current encoding.*
- void [HPDF\\_RoundedCornerRectangle](#) (HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, HPDF\_REAL rad)

*Draw rectangle with rounded corner.*
- void [hpdfctl\\_stroke\\_grid](#) (HPDF\_Doc pdf, HPDF\_Page page)
- void [hpdfctl\\_table\\_widget\\_letter\\_buttons](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, HPDF\_RGBColor on\_color, HPDF\_RGBColor off\_color, HPDF\_RGBColor on\_background, HPDF\_RGBColor off\_background, HPDF\_REAL fsize, const char \*letters, \_Bool \*state)

*Display an array of letters as a table where each letter is its own "mini" cell and surrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different font and fac colors.*
- void [hpdfctl\\_widget\\_slide\\_button](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, \_Bool state)

*Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.*
- void [hpdfctl\\_widget\\_hbar](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, HPDF\_RGBColor color, float val, \_Bool hide\_val)

*Draw a horizontal partially filled bar to indicate an analog (percentage) value.*
- void [hpdfctl\\_widget\\_segment\\_hbar](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, size\_t num\_segments, HPDF\_RGBColor on\_color, double val\_percent, \_Bool hide\_val)

*Draw a horizontal segment meter that can be used to visualize a discrete value.*
- void [hpdfctl\\_widget\\_strength\\_meter](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, size\_t num\_segments, HPDF\_RGBColor on\_color, size\_t num\_on\_segments)

*Draw a phone strength meter.*

## Variables

- int [hpdfctl\\_err\\_code](#)

*Stores the last generated error code.*
- int [hpdfctl\\_err\\_row](#)

*The row where the last error was generated.*
- int [hpdfctl\\_err\\_col](#)

*The column where the last error was generated.*

### 15.7.1 Detailed Description

Header file for libhpdftbl.

#### Author

Johan Persson ( [johan162@gmail.com](mailto:johan162@gmail.com) )

Copyright (C) 2022 Johan Persson

#### See also

LICENSE

Released under the MIT License

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.

### 15.7.2 Macro Definition Documentation

#### 15.7.2.1 \_HPDFTBL\_SET\_ERR

```
#define _HPDFTBL_SET_ERR(  
    t,  
    err,  
    r,  
    c ) do {hpdftbl_err_code=err;hpdftbl_err_row=r;hpdftbl_err_col=c; if(hpdftbl_err_handler){hpdftbl_err_handler(t,r,c,err);}} while(0)
```

Call the error handler with specified error code and table row, col where error occurred.

#### Parameters

<i>t</i>	Table handler
<i>err</i>	Error code
<i>r</i>	Row where error occurred
<i>c</i>	Column where error occurred

### 15.7.2.2 DEFAULT\_AUTO\_VBOTTOM\_MARGIN\_FACTOR

```
#define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5
```

The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size.

The margin is calculated as:

```
bottom_margin = fontsize * AUTO_VBOTTOM_MARGIN_FACTOR
```

See also

`hpdfctl_set_bottom_vmargin_bottom()`

### 15.7.2.3 hpdfctl\_cm2dpi

```
#define hpdfctl_cm2dpi(  
    c ) (( (HPDF_REAL) (c) ) / 2.54 * 72)
```

Convert cm to dots using the default resolution (72 DPI)

Parameters

<code>c</code>	Measure in cm
----------------	---------------

Returns

HPDF\_REAL Converted value in dots

Examples

[example01.c](#), [tut\\_ex01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex13\\_1.c](#), [tut\\_ex13\\_2.c](#), [tut\\_ex14.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

## 15.7.3 Typedef Documentation

### 15.7.3.1 hpdf\_text\_style\_t

```
typedef struct text_style hpdf_text_style_t
```

Specification of a text style.

This structure collects the basic properties for a text string (font, color, background, horizontal alignment)

### 15.7.3.2 `hpdftbl_callback_t`

```
typedef void(* hpdftbl_callback_t) (hpdftbl_t)
```

Callback type for optional post processing when constructing table from a data array.

Type for generic table callback used when constructing a table from data. This can be used to perform any potential table manipulation. The callback happens after the table has been fully constructed and just before it is stroked.

See also

[hpdftbl\\_stroke\\_from\\_data\(\)](#)

### 15.7.3.3 `hpdftbl_canvas_callback_t`

```
typedef void(* hpdftbl_canvas_callback_t) (HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_REAL, HPDF_REAL, HPDF_REAL, HPDF_REAL)
```

Type specification for the table canvas callback.

A canvas callback, if specified, is called for each cell before the content is stroked. The callback will be given the bounding box for the cell (x,y,width,height) in addition to the row and column the cell has.

See also

[hpdftbl\\_set\\_canvas\\_cb\(\)](#)

### 15.7.3.4 `hpdftbl_cell_spec_t`

```
typedef struct hpdftbl_cell_spec hpdftbl_cell_spec_t
```

Used in data driven table creation.

A table can be specified by creating a array of this structure together with the `hpdftbl_spec_t` structure. The array should have one entry for each cell in the table.

See also

[hpdftbl\\_stroke\\_from\\_data\(\)](#)

### 15.7.3.5 `hpdftbl_cell_t`

```
typedef struct hpdftbl_cell hpdftbl_cell_t
```

Type definition for the cell structure.

This is an internal structure that represents an individual cell in the table.

### 15.7.3.6 hpdfctl\_content\_callback\_t

```
typedef char *(* hpdfctl_content_callback_t) (void *, size_t, size_t)
```

Type specification for the table content callback.

The content callback is used to specify the textual content in a cell and is an alternative method to specifying the content to be displayed.

See also

[hpdfctl\\_set\\_content\\_cb\(\)](#)

### 15.7.3.7 hpdfctl\_content\_style\_callback\_t

```
typedef _Bool(* hpdfctl_content_style_callback_t) (void *, size_t, size_t, char *content, hpdf\_text\_style\_t *)
```

Type specification for the content style.

The content callback is used to specify the textual style in a cell and is an alternative method to specifying the style of content to be displayed.

See also

[hpdfctl\\_set\\_content\\_style\\_cb\(\)](#)

### 15.7.3.8 hpdfctl\_error\_handler\_t

```
typedef void(* hpdfctl_error_handler_t) (hpdfctl\_t, int, int, int)
```

Type for error handler function.

The error handler (of set) will be called if the table library discovers an error condition

See also

[hpdfctl\\_set\\_errhandler\(\)](#)

### 15.7.3.9 hpdfctl\_grid\_style\_t

```
typedef struct grid\_style hpdfctl\_grid\_style\_t
```

Specification for table grid lines.

Contains line properties used when stroking a grid line

### 15.7.3.10 `hpdfctl_line_dashstyle_t`

```
typedef enum hpdfctl_dashstyle hpdfctl_line_dashstyle_t
```

Possible line dash styles for grid lines.

In the illustration of the patterns "x"=solid and "\_"=space.

For each pattern we show two full cycles which should give a good visual indication of the different patterns.

### 15.7.3.11 `hpdfctl_spec_t`

```
typedef struct hpdfctl_spec hpdfctl_spec_t
```

Used in data driven table creation.

This is used together with an array of cell specification `hpdfctl_cell_spec_t` to specify the layout of a table.

### 15.7.3.12 `hpdfctl_t`

```
typedef struct hpdfctl* hpdfctl_t
```

Table handle is a pointer to the `hpdfctl` structure.

This is the basic table handle used in almost all API calls. A table reference is returned when a table is created.

See also

[hpdfctl\\_create\(\)](#)

### 15.7.3.13 `hpdfctl_text_align_t`

```
typedef enum hpdfctl_text_align hpdfctl_text_align_t
```

Enumeration for horizontal text alignment.

See also

[hpdfctl\\_set\\_header\\_halign\(\)](#)

[hpdfctl\\_set\\_title\\_halign\(\)](#)

[hpdfctl\\_text\\_align](#)



### 15.7.3.14 hpdftbl\_theme\_t

```
typedef struct hpdftbl_theme hpdftbl_theme_t
```

Define a set of styles into a table theme.

Contains all information about the styles of various elements in the table that together make up the table style

## 15.7.4 Enumeration Type Documentation

### 15.7.4.1 hpdftbl\_dashstyle

```
enum hpdftbl_dashstyle
```

Possible line dash styles for grid lines.

In the illustration of the patterns "x"=solid and "\_"=space.

For each pattern we show two full cycles which should give a good visual indication of the different patterns.

Enumerator

LINE_SOLID	Solid line
LINE_DOT1	Dotted line variant 1 "x_x_x_"
LINE_DOT2	Dotted line variant 2 "x_x_x_x_"
LINE_DOT3	Dotted line variant 3 "x_x_x_x_x_"
LINE_DASH1	Dashed line variant 1 "xx_xx_xx_"
LINE_DASH2	Dashed line variant 2 "xx_xx_xx_x_"
LINE_DASH3	Dashed line variant 3 "xxxx_xxxx_xxxx_"
LINE_DASH4	Dashed line variant 4 "xxxx_xxxx_xxxx_x_"
LINE_DASHDOT1	Dashed-dot line variant 1 "xxxxx_xx_xxxxx_xx_xxxxx_xx_"
LINE_DASHDOT2	Dashed-dot line variant 1 "xxxxxxx_xxx_xxxxxxxx_xxx_xxxxxxxx_xxx_"

### 15.7.4.2 hpdftbl\_text\_align

```
enum hpdftbl_text_align
```

Enumeration for horizontal text alignment.

See also

[hpdftbl\\_set\\_header\\_halign\(\)](#)  
[hpdftbl\\_set\\_title\\_halign\(\)](#)  
[hpdftbl\\_text\\_align](#)

## Enumerator

LEFT	Left test alignment
CENTER	Center test alignment
RIGHT	Right test alignment

## 15.7.5 Function Documentation

### 15.7.5.1 HPDF\_RoundedCornerRectangle()

```
void HPDF_RoundedCornerRectangle (
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    HPDF_REAL width,
    HPDF_REAL height,
    HPDF_REAL rad )
```

Draw rectangle with rounded corner.

Draw a rectangle with rounded corner with the current line width, color. The rectangle will not be stroked.

## Parameters

<i>page</i>	Page handle
<i>xpos</i>	Lower left x-position of rectangle
<i>ypos</i>	Lower left y-position of rectangle
<i>width</i>	Width of rectangle
<i>height</i>	Height of rectangle
<i>rad</i>	Radius of corners

Referenced by [hpdfbl\\_widget\\_slide\\_button\(\)](#).

### 15.7.5.2 hpdfbl\_apply\_theme()

```
int hpdfbl_apply_theme (
    hpdfbl_t t,
    hpdfbl_theme_t * theme )
```

Apply a specified theme to a table.

The default table theme can be retrieved with [hpdfbl\\_get\\_default\\_theme\(\)](#)

## Parameters

<i>t</i>	Table handle
<i>theme</i>	Theme reference

## Returns

0 on success, -1 on failure

## See also

[hpdftbl\\_get\\_default\\_theme\(\)](#)

Referenced by [hpdftbl\\_create\\_title\(\)](#), and [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.3 hpdftbl\_clear\_spanning()

```
int hpdftbl_clear_spanning (
    hpdftbl_t t )
```

Clear all cell spanning.

Reset all spanning cells to no spanning

## Parameters

<i>t</i>	Table handle
----------	--------------

## Returns

0 on success, -1 on failure

## See also

[hpdftbl\\_set\\_cellspan\(\)](#)

### 15.7.5.4 hpdftbl\_create()

```
hpdftbl_t hpdftbl_create (
    size_t rows,
    size_t cols )
```

Create a new table with no title.

Create a new table structure. This is the basic handler needed for most other API functions.

**Parameters**

<i>rows</i>	Number of rows
<i>cols</i>	Number of columns

**Returns**

A handle to a table, NULL in case of OOM

**15.7.5.5 hpdftbl\_create\_title()**

```
hpdftbl_t hpdftbl_create_title (
    size_t rows,
    size_t cols,
    char * title )
```

Create a new table with title top row.

Create a new table structure. This is the basic handler needed for most other API functions.

**Parameters**

<i>rows</i>	Number of rows
<i>cols</i>	Number of columns
<i>title</i>	Title of table

**Returns**

A handle to a table, NULL in case of OOM

Referenced by [hpdftbl\\_create\(\)](#), and [hpdftbl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.6 hpdftbl\_default\_table\_error\_handler()**

```
void hpdftbl_default_table_error_handler (
    hpdftbl_t t,
    int r,
    int c,
    int err )
```

A basic default table error handler.

This error handler is used as a callback that outputs the error to stderr in human readable format and quits the process.

## Parameters

<i>t</i>	Table where the error happened (can be NULL)
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>err</i>	The error code

## See also

[hpdftbl\\_set\\_errhandler\(\)](#)

### 15.7.5.7 hpdftbl\_destroy()

```
int hpdftbl_destroy (
    hpdftbl_t t )
```

Destroy a table and free all memory.

Destroy a table previous created with `table_create()`, It is the calling routines responsibility not to access `t` again.

## Parameters

<i>t</i>	Handle to table
----------	-----------------

## Returns

0 on success, -1 on failure

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.8 hpdftbl\_destroy\_theme()

```
int hpdftbl_destroy_theme (
    hpdftbl_theme_t * theme )
```

Destroy existing theme structure and free memory.

Free all memory allocated by a theme

## Parameters

<i>theme</i>	The theme to free
--------------	-------------------

**Returns**

-1 for error , 0 for success

**Examples**

[example01.c](#).

Referenced by [hpdfdbl\\_create\\_title\(\)](#).

**15.7.5.9 hpdfdbl\_encoding\_text\_out()**

```
int hpdfdbl_encoding_text_out (
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    char * text )
```

Stroke text with current encoding.

Utility function to stroke text with character encoding. It is the calling routines responsibility to enclose text in a HPDF\_Page\_BeginText() / HPDF\_Page\_EndText()

**Parameters**

<i>page</i>	Page handle
<i>xpos</i>	X coordinate
<i>ypos</i>	Y coordinate
<i>text</i>	Text to print

**Returns**

-1 on error, 0 on success

**15.7.5.10 hpdfdbl\_get\_anchor\_top\_left()**

```
_Bool hpdfdbl_get_anchor_top_left (
    void )
```

Get stroking anchor point.

Get anchor point for table positioning. By default the top left is used.

**See also**

[hpdfdbl\\_set\\_anchor\\_top\\_left](#)

**Returns**

TRUE if anchor is top left, FALSE otherwise

### 15.7.5.11 hpdftbl\_get\_default\_theme()

```
hpdftbl_theme_t * hpdftbl_get_default_theme (
    void )
```

Return the default theme.

Create and return a theme corresponding to the default table theme. It is the calling functions responsibility to call [hpdftbl\\_destroy\\_theme\(\)](#) to free the allocated memory. The default theme is a good starting point to just make minor modifications without having to define all elements.

#### Returns

A new theme initialized to the default settings

#### See also

[hpdftbl\\_apply\\_theme\(\)](#)

#### Examples

[example01.c](#).

Referenced by [hpdftbl\\_create\\_title\(\)](#).

### 15.7.5.12 hpdftbl\_get\_errstr()

```
const char * hpdftbl_get_errstr (
    int err )
```

Translate a table error code to a human readable string.

The function returns a pointer to a static string that cannot be modified. It will translate both internal table error messages as well as generic HPDF library error codes.

#### Parameters

<i>err</i>	The error code to be translated
------------	---------------------------------

#### Returns

Static pointer to string for valid error code, NULL otherwise

#### See also

[hpdftbl\\_hpdf\\_get\\_errstr\(\)](#)

Referenced by [hpdftbl\\_default\\_table\\_error\\_handler\(\)](#), and [hpdftbl\\_get\\_last\\_errcode\(\)](#).

### 15.7.5.13 `hpdftbl_get_last_auto_height()`

```
int hpdftbl_get_last_auto_height (
    HPDF_REAL * height )
```

Get the height calculated for the last constructed table.

Get the last automatically calculated height when stroking a table. (The height will be automatically calculated if it was specified as 0)

#### Parameters

<i>height</i>	Returned height
---------------	-----------------

#### Returns

-1 on error, 0 if successful

### 15.7.5.14 `hpdftbl_get_last_errcode()`

```
int hpdftbl_get_last_errcode (
    const char ** errstr,
    int * row,
    int * col )
```

Return last error code.

Return last error code. if *errstr* is not NULL a human readable string describing the error will be copied to the string. The error code will be reset after call.

#### Parameters

<i>errstr</i>	A string buffer where the error string is written to
<i>row</i>	The row where the error was found
<i>col</i>	The col where the error was found

#### Returns

The last error code

### 15.7.5.15 `hpdftbl_hpdf_get_errstr()`

```
const char * hpdftbl_hpdf_get_errstr (
    const HPDF_STATUS err_code )
```

Function to return a human readable error string for an error code from Core HPDF library.

The various error codes given by the HPDF library can be translated back to a string by the usage of this function. The function will return a pointer to a static string that can not be manipulated.



## Parameters

<i>err_code</i>	The error code
-----------------	----------------

## Returns

A pointer to an error string, NULL if the error code is invalid

## See also

[hpdftbl\\_get\\_errstr\(\)](#)

## Examples

[example01.c](#), [tut\\_ex01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex13\\_1.c](#), [tut\\_ex13\\_2.c](#), [tut\\_ex14.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

## 15.7.5.16 hpdftbl\_set\_anchor\_top\_left()

```
void hpdftbl_set_anchor_top_left (
    const _Bool anchor )
```

Switch stroking anchor point.

Set anchor point for table positioning. By default the top left is used as anchor. Calling this function with FALSE can sets the anchor to bottom left instead.

## Parameters

<i>anchor</i>	Set to TRUE to use top left as anchor, FALSE for bottom left
---------------	--

## 15.7.5.17 hpdftbl\_set\_background()

```
int hpdftbl_set_background (
    hpdftbl_t t,
    HPDF_RGBColor background )
```

Set table background color.

## Parameters

<i>t</i>	Table handle
<i>background</i>	Background color

**Returns**

0 on success, -1 on failure

**15.7.5.18 hpdftbl\_set\_bottom\_vmargin\_factor()**

```
void hpdftbl_set_bottom_vmargin_factor (
    hpdftbl_t t,
    HPDF_REAL f )
```

The margin from the bottom of the cell to the baseline of the text is calculated as a fraction of the font size. The margin is calculated as:

```
bottom_margin = fontsize * f
```

The default margin is specified by the define [DEFAULT\\_AUTO\\_VBOTTOM\\_MARGIN\\_FACTOR](#)

**Parameters**

<i>t</i>	Table handle
<i>f</i>	Bottom margin factor

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.7.5.19 hpdftbl\_set\_canvas\_cb()**

```
int hpdftbl_set_canvas_cb (
    hpdftbl_t t,
    hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set cell canvas callback. This callback gets called for each cell in the table. The purpose is to allow the client to add dynamic content to the specified cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell. To set the canvas callback only for a specific cell use the [hpdftbl\\_set\\_cell\\_canvas\\_cb\(\)](#) function

**Parameters**

<i>t</i>	Table handle
<i>cb</i>	Callback function

**Returns**

-1 on failure, 0 otherwise

See also

[hpdftbl\\_set\\_cell\\_canvas\\_cb\(\)](#)

### 15.7.5.20 hpdftbl\_set\_cell()

```
int hpdftbl_set_cell (
    hpdftbl_t t,
    int r,
    int c,
    char * label,
    char * content )
```

Set content for specific cell.

Set label and content for a specific cell. If the specified cell is part of another cells spanning an error occurs (returns -1),

#### Parameters

<i>t</i>	Table handle
<i>r</i>	Row
<i>c</i>	Column
<i>label</i>	Label
<i>content</i>	Text content

#### Returns

-1 on error, 0 if successful

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.21 hpdftbl\_set\_cell\_canvas\_cb()

```
int hpdftbl_set_cell_canvas_cb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    hpdftbl_canvas_callback_t cb )
```

Set cell canvas callback.

Set a canvas callback for an individual cell. This will override the table canvas callback. The canvas callback is called with arguments that give the bounding box for the cell. In that way a callback function may draw arbitrary graphic in the cell. The callback is made before the cell border and content is drawn making it possible to for example add a background color to individual cells. The callback function will receive the Table tag, the row and column, the x, y position of the lower left corner of the table and the width and height of the cell.

**Parameters**

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb</i>	Callback function

**Returns**

-1 on failure, 0 otherwise

**See also**

[hpdfdbl\\_canvas\\_callback\\_t](#)

[hpdfdbl\\_set\\_canvas\\_cb\(\)](#)

Referenced by [hpdfdbl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.22 hpdfdbl\_set\_cell\_content\_cb()**

```
int hpdfdbl_set_cell_content_cb (
    hpdfdbl_t t,
    size_t r,
    size_t c,
    hpdfdbl_content_callback_t cb )
```

Set cell content callback.

Set a content callback for an individual cell. This will override the table content callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

**Parameters**

<i>t</i>	Table handle
<i>cb</i>	Callback function
<i>r</i>	Cell row
<i>c</i>	Cell column

**Returns**

-1 on failure, 0 otherwise

**See also**

[hpdfdbl\\_set\\_content\\_cb\(\)](#)

Referenced by [hpdfdbl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.23 hpdftbl\_set\_cell\_content\_style()**

```
int hpdftbl_set_cell_content_style (
    hpdftbl_t t,
    size_t r,
    size_t c,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the font style for content of specified cell.

SSet the font style for content of specified cell. This will override the global cell content setting.

**Parameters**

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_content\\_style\(\)](#)  
[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

Referenced by [hpdftbl\\_set\\_col\\_content\\_style\(\)](#), and [hpdftbl\\_set\\_row\\_content\\_style\(\)](#).

**15.7.5.24 hpdftbl\_set\_cell\_content\_style\_cb()**

```
int hpdftbl_set_cell_content_style_cb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    hpdftbl_content_style_callback_t cb )
```

Set cell specific callback to specify cell content style.

Set callback to format the style for the specified cell

**Parameters**

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb</i>	Callback function

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_content\\_style\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.25 hpdftbl\_set\_cell\_label\_cb()**

```
int hpdftbl_set_cell_label_cb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    hpdftbl_content_callback_t cb )
```

Set cell label callback.

Set a label callback for an individual cell. This will override the table label callback. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

**Parameters**

<i>t</i>	Table handle
<i>cb</i>	Callback function
<i>r</i>	Cell row
<i>c</i>	Cell column

**Returns**

-1 on failure, 0 otherwise

**See also**

[hpdftbl\\_set\\_label\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.26 hpdfctl\_set\_cellspan()

```
int hpdfctl_set_cellspan (
    hpdfctl_t t,
    size_t r,
    size_t c,
    size_t rowspan,
    size_t colspan )
```

Set cell spanning.

Set row and column spanning for a cell, an expanded cell is referenced via the position of it's top-left cell

#### Parameters

<i>t</i>	Table handle
<i>r</i>	Row
<i>c</i>	Column
<i>rowspan</i>	Row span
<i>colspan</i>	Column span

#### Returns

-1 on error, 0 if successful

#### See also

[hpdfctl\\_clear\\_spanning\(\)](#)

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.27 hpdfctl\_set\_col\_content\_style()

```
int hpdfctl_set_col_content_style (
    hpdfctl_t t,
    size_t c,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the font style for an entire column of cells.

Set font options for the specified column of cells. This will override the global cell content setting.

#### Parameters

<i>t</i>	Table handle
<i>c</i>	Column to affect
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_content\\_style\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

**15.7.5.28 hpdftbl\_set\_colwidth\_percent()**

```
int hpdftbl_set_colwidth_percent (
    hpdftbl_t t,
    size_t c,
    float w )
```

Set column width as percentage of overall table width.

Specify column width as percentage of total column width. Note that this will only take effect if the table has an overall width specified when stroked. Too avoid errors one column should be left unspecified to let the library use whatever space is left for that column.

**Parameters**

<i>t</i>	Table handle
<i>c</i>	Column to set width of first column has index 0
<i>w</i>	Width as percentage in range [0.0, 100.0]

**Returns**

0 on success, -1 on failure

**15.7.5.29 hpdftbl\_set\_content()**

```
int hpdftbl_set_content (
    hpdftbl_t t,
    char ** content )
```

Set the content for the table.

Set content for all cells. It is the calling functions responsibility that the content array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r \* num\_cols + c) where num\_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with strdup() so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N\*M) entries.

Another way to specify the content is to use the callback mechanism. By setting up a content callback function that returns the content for a cell.



## Parameters

<i>t</i>	Table handle
<i>content</i>	A one dimensional string array of content string

## Returns

-1 on error, 0 if successful

## See also

[hpdftbl\\_set\\_content\\_callback\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_callback\(\)](#)

**15.7.5.30 hpdftbl\_set\_content\_cb()**

```
int hpdftbl_set_content_cb (
    hpdftbl_t t,
    hpdftbl_content_callback_t cb )
```

Set table content callback.

This callback gets called for each cell in the table and the returned string will be used as the content. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the content will be set to the content specified with the direct content setting. The callback function will receive the Table tag and the row and column for the cell the callback is made for.

## Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function

## Returns

-1 for error , 0 otherwise

## See also

[hpdftbl\\_set\\_cell\\_content\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.31 `hpdftbl_set_content_style()`

```
int hpdftbl_set_content_style (
    hpdftbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set style for text content.

Set style options for cell content (font, color, background). This will be applied for all cells in the table. If a style callback have been specified for either the table or a cell that style take precedence.

#### Parameters

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

#### Returns

-1 on error, 0 if successful

#### See also

[hpdftbl\\_set\\_cell\\_content\\_style\(\)](#)  
[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

### 15.7.5.32 `hpdftbl_set_content_style_cb()`

```
int hpdftbl_set_content_style_cb (
    hpdftbl_t t,
    hpdftbl_content_style_callback_t cb )
```

Set callback to specify cell content style.

Set callback to format the style for cells in the table. If a cell has its own content style callback that callback will override the generic table callback.

#### Parameters

<i>t</i>	Table handle
<i>cb</i>	Callback function

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.33 hpdftbl\_set\_errhandler()**

```
hpdftbl_error_handler_t hpdftbl_set_errhandler (  
    hpdftbl_error_handler_t err_handler )
```

Specify errhandler for the table routines.

Note: The library provides a basic default error handler that can be used,

**Parameters**

<i>err_handler</i>	
--------------------	--

**Returns**

The old error handler or NULL if non exists

**See also**

[hpdftbl\\_default\\_table\\_error\\_handler\(\)](#)

**15.7.5.34 hpdftbl\_set\_header\_halign()**

```
int hpdftbl_set_header_halign (  
    hpdftbl_t t,  
    hpdftbl_text_align_t align )
```

Set table header horizontal text align.

**Parameters**

<i>t</i>	Table handle
<i>align</i>	Alignment

**Returns**

0 on success, -1 on failure

Referenced by [hpdfctl\\_apply\\_theme\(\)](#).

**15.7.5.35 hpdfctl\_set\_header\_style()**

```
int hpdfctl_set_header_style (
    hpdfctl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Specify style for table header row.

Set the font properties and background for the header row which is the top row if enabled. The header row will be automatically enabled after calling this function. The header can be enabled/disabled separately with [hpdfctl\\_use\\_header\(\)](#)

**Parameters**

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Font color
<i>background</i>	Cell background color

**Returns**

0 on success, -1 on failure

**See also**

[hpdfctl\\_use\\_header\(\)](#)

Referenced by [hpdfctl\\_apply\\_theme\(\)](#).

**15.7.5.36 hpdfctl\_set\_inner\_grid\_style()**

```
int hpdfctl_set_inner_grid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_line_dashstyle_t dashstyle )
```

Set inner border grid style.

This is a shortform to set both the vertical and horizontal gridline style with one call.

## Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

## Returns

0 on success, -1 on failure

## See also

[hpdfctl\\_set\\_inner\\_hgrid\\_style\(\)](#), [hpdfctl\\_set\\_inner\\_vgrid\\_style\(\)](#), [hpdfctl\\_set\\_outer\\_grid\\_style\(\)](#)

**15.7.5.37 hpdfctl\_set\_inner\_hgrid\_style()**

```
int hpdfctl_set_inner_hgrid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_line_dashstyle_t dashstyle )
```

Set inner horizontal border grid style.

## Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

## Returns

0 on success, -1 on failure

## See also

[hpdfctl\\_set\\_inner\\_grid\\_style\(\)](#), [hpdfctl\\_set\\_inner\\_vgrid\\_style\(\)](#)

Referenced by [hpdfctl\\_apply\\_theme\(\)](#), and [hpdfctl\\_set\\_inner\\_grid\\_style\(\)](#).

### 15.7.5.38 `hpdftbl_set_inner_tgrid_style()`

```
int hpdftbl_set_inner_tgrid_style (
    hpdftbl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdftbl_line_dashstyle_t dashstyle )
```

Set inner horizontal top border grid style.

This would be the gridline just below the header row.

#### Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

#### Returns

0 on success, -1 on failure

#### See also

[hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#)

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

### 15.7.5.39 `hpdftbl_set_inner_vgrid_style()`

```
int hpdftbl_set_inner_vgrid_style (
    hpdftbl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdftbl_line_dashstyle_t dashstyle )
```

Set inner vertical border grid style.

#### Parameters

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#), [hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#)

Referenced by [hpdftbl\\_apply\\_theme\(\)](#), and [hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#).

**15.7.5.40 hpdftbl\_set\_label\_cb()**

```
int hpdftbl_set_label_cb (
    hpdftbl_t t,
    hpdftbl_content_callback_t cb )
```

Set table label callback.

Set label callback. This callback gets called for each cell in the table and the returned string will be used as the label. The string will be duplicated so it is safe for a client to reuse the string space. If NULL is returned from the callback then the label will be set to the content specified with the direct label setting. The callback function will receive the Table tag and the row and column

**Parameters**

<i>t</i>	Table handle
<i>cb</i>	Callback function

**Returns**

-1 on failure, 0 otherwise

**See also**

[hpdftbl\\_content\\_callback\\_t](#)

[hpdftbl\\_set\\_cell\\_label\\_cb\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.41 hpdftbl\_set\_label\_style()**

```
int hpdftbl_set_label_style (
    hpdftbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the style for labels in the entire table.

Set font, color and background options for cell labels. If a style callback have been specified for either the table or a cell that style take precedence.

**Parameters**

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

**Returns**

-1 on error, 0 if successful

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.7.5.42 hpdftbl\_set\_labels()**

```
int hpdftbl_set_labels (
    hpdftbl_t t,
    char ** labels )
```

Set the text for the cell labels.

Set labels for all the cell. It is the calling functions responsibility that the labels array is big enough to cover the entire table. The string array corresponds to a flattened 2-d array and the label for cell (r,c) is calculated as (r \* num\_cols + c) where num\_cols is the number of columns in the table.

It is allowed to specify NULL as placeholder for empty labels. The actual text in the table will be allocated with `strdup()` so it is safe to free the memory for the labels after the call to this function. Please note that even if the table contains spanning cells the content data must include empty data for covered cells. For a N x M table the data must have (N\*M) entries.

**Parameters**

<i>t</i>	Table handle
<i>labels</i>	A one dimensional string array of labels

**Returns**

-1 on error, 0 if successful

**See also**

[hpdftbl\\_set\\_cell\\_label\\_cb\(\)](#)

[hpdftbl\\_set\\_label\\_cb\(\)](#)



**15.7.5.43 hpdftbl\_set\_min\_rowheight()**

```
int hpdftbl_set_min_rowheight (
    hpdftbl_t t,
    float h )
```

Set the minimum row height in the table.

The row height is normally calculated based on the font size and if labels are displayed or not. However, it is not possible for the table to know the height of specific widgets (for example) without a two-pass table drawing algorithm.

To handle those odd cases when the calculated height is not sufficient a manual minimum height can be specified.

**Parameters**

<i>t</i>	Table handler
<i>h</i>	The minimum height (in points). If specified as 0 the min height will have no effect.

**Returns**

0 on success, -1 on failure

**15.7.5.44 hpdftbl\_set\_outer\_grid\_style()**

```
int hpdftbl_set_outer_grid_style (
    hpdftbl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdftbl_line_dashstyle_t dashstyle )
```

Set outer border grid style.

**Parameters**

<i>t</i>	Table handle
<i>width</i>	Line width (in pt)
<i>color</i>	Line color
<i>dashstyle</i>	Line dash style

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_inner\\_grid\\_style\(\)](#)

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

### 15.7.5.45 `hpdftbl_set_row_content_style()`

```
int hpdftbl_set_row_content_style (
    hpdftbl_t t,
    size_t r,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the style for an entire row of cells.

Set font options for the specified row of cells. This will override the global cell content.

#### Parameters

<i>t</i>	Table handle
<i>r</i>	Row to affect
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

#### Returns

0 on success, -1 on failure

#### See also

[hpdftbl\\_set\\_content\\_style\(\)](#)

[hpdftbl\\_set\\_cell\\_content\\_style\\_cb\(\)](#)

### 15.7.5.46 `hpdftbl_set_tag()`

```
int hpdftbl_set_tag (
    hpdftbl_t t,
    void * tag )
```

Set an optional tag for the table.

Set an optional tag in the table. The tag can be a pointer to anything. The tag is passed as the first argument in the various callbacks and can be used to supply table specific information or identify a specific table in the case the same callback is used for multiple tables.

#### Parameters

<i>t</i>	The table handle
<i>tag</i>	The tag (pointer to any object)

**Returns**

0 on success, -1 on failure

**15.7.5.47 hpdftbl\_set\_text\_encoding()**

```
void hpdftbl_set_text_encoding (
    char * target,
    char * source )
```

Determine text source encoding.

The default HPDF encoding is a standard PDF encoding. The problem with that is that now almost 100% of all code is written in UTF-8 encoding and trying to print text strings with accented characters will simply not work. For example the default encoding assumes that strings are given in UTF-8 and sets the target to ISO8859-4 which includes northern europe accented characters. The conversion is internally handled by the standard `iconv()` routines.

**Parameters**

<i>target</i>	The target encoding. See HPDF documentation for supported encodings.
<i>source</i>	The source encodings, i.e. what encodings are sth strings in the source specified in.

**15.7.5.48 hpdftbl\_set\_title()**

```
int hpdftbl_set_title (
    hpdftbl_t t,
    char * title )
```

Set table title.

Set table title. A title will occupy a separate row above the table that is not included in the row count. A table is enabled when the table text is `<> NULL` and disabled when the title text is `== NULL`.

**Parameters**

<i>t</i>	Table handle
<i>title</i>	Title string

**Returns**

0 on success, -1 on failure

**See also**

[hpdftbl\\_set\\_title\\_style\(\)](#)

[hpdftbl\\_set\\_title\\_halign\(\)](#)

### 15.7.5.49 `hpdfdbl_set_title_halign()`

```
int hpdfdbl_set_title_halign (
    hpdfdbl_t t,
    hpdfdbl_text_align_t align )
```

Set horizontal alignment for table title.

#### Parameters

<i>t</i>	Table handle
<i>align</i>	Alignment

#### Returns

0 on success, -1 on failure

#### See also

[hpdfdbl\\_set\\_title\(\)](#)

[hpdfdbl\\_set\\_title\\_style\(\)](#)

Referenced by [hpdfdbl\\_apply\\_theme\(\)](#).

### 15.7.5.50 `hpdfdbl_set_title_style()`

```
int hpdfdbl_set_title_style (
    hpdfdbl_t t,
    char * font,
    HPDF_REAL fsize,
    HPDF_RGBColor color,
    HPDF_RGBColor background )
```

Set the table title style.

Set font options for title

#### Parameters

<i>t</i>	Table handle
<i>font</i>	Font name
<i>fsize</i>	Font size
<i>color</i>	Color
<i>background</i>	Background color

#### Returns

0 on success, -1 on failure

See also

[hpdftbl\\_set\\_title\(\)](#)

[hpdftbl\\_set\\_title\\_halign\(\)](#)

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

#### 15.7.5.51 hpdftbl\_set\_zebra()

```
int hpdftbl_set_zebra (
    hpdftbl_t t,
    _Bool use,
    int phase )
```

Parameters

<i>t</i>	Table handle
<i>use</i>	TRUE=Use Zebra, FALSE=Don't use zebra
<i>phase</i>	0=Start with color 1, 1=Start with color 1

Returns

0 on successes -1 on failure

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

#### 15.7.5.52 hpdftbl\_set\_zebra\_color()

```
int hpdftbl_set_zebra_color (
    hpdftbl_t t,
    HPDF_RGBColor z1,
    HPDF_RGBColor z2 )
```

Specify first and second color for a zebra grid table.

By default the colors start with *z1* color. To have the top row (below any potential header row) instead start with *z2* specify *phase*=1 in the [hpdftbl\\_set\\_zebra\(\)](#) function.

Parameters

<i>t</i>	Table handle
<i>z1</i>	Color 1
<i>z2</i>	Color 2

**Returns**

0 on successes -1 on failure

Referenced by [hpdftbl\\_apply\\_theme\(\)](#).

**15.7.5.53 hpdftbl\_stroke()**

```
int hpdftbl_stroke (
    HPDF_Doc pdf,
    const HPDF_Page page,
    hpdftbl_t t,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    HPDF_REAL height )
```

Stroke the table.

Stroke the table at the specified position and size. The position is by default specified as the upper left corner of the table. Use the `hpdftbl_set_origin_top_left(FALSE)` to use the bottom left of the table as reference point.

**Parameters**

<i>pdf</i>	The HPDF document handle
<i>page</i>	The HPDF page handle
<i>t</i>	Table handle
<i>xpos</i>	x position for table, bottom left corner
<i>ypos</i>	y position for table, bottom left corner
<i>width</i>	width of table
<i>height</i>	height of table. If the height is specified as 0 it will be automatically calculated. The calculated height can be retrieved after the table has been stroked by a call to <a href="#">hpdftbl_get_last_auto_height()</a>

**Returns**

-1 on error, 0 if successful

**See also**

[hpdftbl\\_get\\_last\\_auto\\_height\(\)](#)

[hpdftbl\\_stroke\\_from\\_data\(\)](#)

Referenced by [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.7.5.54 hpdftbl\_stroke\_from\_data()

```
int hpdftbl_stroke_from_data (
    HPDF_Doc pdf_doc,
    HPDF_Page pdf_page,
    hpdftbl_spec_t * tbl_spec,
    hpdftbl_theme_t * theme )
```

Construct the table from a array specification.

Create and stroke a table specified by a data structure. This makes it easier to separate the view of the data from the model which provides the data. The intended use case is that the data structure specifies the core layout of the table together with the labels and callback functions to handle the content in each cell. Using this method to create a table also makes it much more maintainable.

#### Parameters

<i>pdf_doc</i>	The PDF overall document
<i>pdf_page</i>	The pageto stroke to
<i>tbl_spec</i>	The table specification
<i>theme</i>	Table theme to be applied

#### Returns

0 on success, -1 on failure

#### See also

[hpdftbl\\_stroke\(\)](#)

### 15.7.5.55 hpdftbl\_stroke\_grid()

```
void hpdftbl_stroke_grid (
    HPDF_Doc pdf,
    HPDF_Page page )
```

Stroke a point grid on specified page to make it easier to position text and tables.

#### Parameters

<i>pdf</i>	Document handle
<i>page</i>	Page handle

#### Examples

[tut\\_ex01.c](#), [tut\\_ex02.c](#), [tut\\_ex02\\_1.c](#), [tut\\_ex03.c](#), [tut\\_ex04.c](#), [tut\\_ex05.c](#), [tut\\_ex06.c](#), [tut\\_ex07.c](#), [tut\\_ex08.c](#), [tut\\_ex09.c](#), [tut\\_ex10.c](#), [tut\\_ex11.c](#), [tut\\_ex12.c](#), [tut\\_ex13\\_1.c](#), [tut\\_ex13\\_2.c](#), [tut\\_ex14.c](#), [tut\\_ex15.c](#), [tut\\_ex15\\_1.c](#), and [tut\\_ex20.c](#).

### 15.7.5.56 hpdftbl\_table\_widget\_letter\_buttons()

```
void hpdftbl_table_widget_letter_buttons (
    HPDF_Doc doc,
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    HPDF_REAL width,
    HPDF_REAL height,
    const HPDF_RGBColor on_color,
    const HPDF_RGBColor off_color,
    const HPDF_RGBColor on_background,
    const HPDF_RGBColor off_background,
    const HPDF_REAL fsize,
    const char * letters,
    _Bool * state )
```

Display an array of letters as a table where each letter is its own "mini" cell and surrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different font and fac colors.

#### Parameters

<i>doc</i>	HPDF document handle
<i>page</i>	HPDF page handle
<i>xpos</i>	X-öosition of cell
<i>ypos</i>	Y-Position of cell
<i>width</i>	Width of cell
<i>height</i>	Height of cell
<i>on_color</i>	The font color in "on" state
<i>off_color</i>	The font color in "off" state
<i>on_background</i>	The face color in "on" state
<i>off_background</i>	The face color in "off" state
<i>fsize</i>	The font size
<i>letters</i>	What letters to have in the boxes
<i>state</i>	What state each boxed letter should be (0=off, 1=on)

#### Examples

[example01.c](#).

### 15.7.5.57 hpdftbl\_use\_header()

```
int hpdftbl_use_header (
    hpdftbl_t t,
    _Bool use )
```



Enable/disable the interpretation of the top row as a header row.

A header row will have a different style and labels will be disabled on this row. In addition the text will be centered vertically and horizontal in the cell.

**Parameters**

<i>t</i>	Table handle
<i>use</i>	TRUE to enable, FALSE to disable

**Returns**

0 on success, -1 on failure

**See also**

[hpdfctl\\_set\\_header\\_style\(\)](#)

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.58 hpdfctl\_use\_labelgrid()**

```
int hpdfctl_use_labelgrid (
    hpdfctl_t t,
    _Bool use )
```

Shorter vertical line to mark labels.

Set the usage of special grid style where the vertical grid only covers the label text and a gap to the next line. Horizontal lines are drawn as usual. The label grid style gives the table a "lighter" look.

**Parameters**

<i>t</i>	Table handle
<i>use</i>	TRUE to use label grid, FALSE o disable it

**Returns**

0 on success, -1 on failure

**See also**

[hpdfctl\\_use\\_labels\(\)](#)

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

#### 15.7.5.59 hpdftbl\_use\_labels()

```
int hpdftbl_use_labels (
    hpdftbl_t t,
    _Bool use )
```

Enable/Disable the use of cell labels.

By default a newly created table will not use cell labels. Enabling labels will also by default enable the special label grid style. To adjust the grid style separately us the [hpdftbl\\_use\\_labelgrid\(\)](#) method.

## Parameters

<i>t</i>	Table handle
<i>use</i>	Set to TRUE for cell labels

## Returns

0 on success, -1 on failure

## See also

[hpdfctl\\_use\\_labelgrid\(\)](#)

Referenced by [hpdfctl\\_stroke\\_from\\_data\(\)](#).

**15.7.5.60 hpdfctl\_widget\_hbar()**

```
void hpdfctl_widget_hbar (
    const HPDF_Doc doc,
    const HPDF_Page page,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    const HPDF_REAL height,
    const HPDF_RGBColor color,
    const float val,
    const _Bool hide_val )
```

Draw a horizontal partially filled bar to indicate an analog (percentage) value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

## Parameters

<i>doc</i>	HPDF Document handle
<i>page</i>	HPDF Page handle
<i>xpos</i>	Lower left x
<i>ypos</i>	Lower left y
<i>width</i>	Width of meter
<i>height</i>	Height of meter
<i>color</i>	Fill color for bar
<i>val</i>	Percentage fill in range [0.0, 100.0]
<i>hide_val</i>	TRUE to hide the value (in percent) at the right end of the entire bar

## Examples

[example01.c](#).

**15.7.5.61 hpdfctl\_widget\_segment\_hbar()**

```
void hpdfctl_widget_segment_hbar (
    const HPDF_Doc doc,
    const HPDF_Page page,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    const HPDF_REAL height,
    const size_t num_segments,
    const HPDF_RGBColor on_color,
    const double val_percent,
    const _Bool hide_val )
```

Draw a horizontal segment meter that can be used to visualize a discrete value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

**Parameters**

<i>doc</i>	HPDF Document handle
<i>page</i>	HPDF Page handle
<i>xpos</i>	Lower left x
<i>ypos</i>	Lower left y
<i>width</i>	Width of meter
<i>height</i>	Height of meter
<i>num_segments</i>	Total number of segments
<i>on_color</i>	Color for "on" segment
<i>val_percent</i>	To what extent should the bars be filled (as a value 0.0 - 1.0)
<i>hide_val</i>	TRUE to hide the value (in percent) at the right end of the entire bar

**Examples**

[example01.c](#), and [tut\\_ex14.c](#).

**15.7.5.62 hpdfctl\_widget\_slide\_button()**

```
void hpdfctl_widget_slide_button (
    HPDF_Doc doc,
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    HPDF_REAL width,
    HPDF_REAL height,
    _Bool state )
```

Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.

This function can not be used directly as a canvas callback since it needs the state of the button as an argument. Instead create a simple canvas callback that determines the wanted state and then just passes on all argument to this widget function.

#### Parameters

<i>doc</i>	HPDF document handle
<i>page</i>	HPDF page handle
<i>xpos</i>	X-Position of cell
<i>ypos</i>	Y-Position of cell
<i>width</i>	Width of cell
<i>height</i>	Height of cell
<i>state</i>	State of button On/Off

#### Examples

[example01.c](#).

#### 15.7.5.63 hpdftbl\_widget\_strength\_meter()

```
void hpdftbl_widget_strength_meter (
    const HPDF_Doc doc,
    const HPDF_Page page,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    const HPDF_REAL height,
    const size_t num_segments,
    const HPDF_RGBColor on_color,
    const size_t num_on_segments )
```

Draw a phone strength meter.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

#### Parameters

<i>doc</i>	HPDF Document handle
<i>page</i>	HPDF Page handle
<i>xpos</i>	Lower left x
<i>ypos</i>	Lower left y
<i>width</i>	Width of meter
<i>height</i>	Height of meter
<i>num_segments</i>	Total number of segments
<i>on_color</i>	Color for "on" segment
<i>num_on_segments</i>	Number of on segments

## Examples

[example01.c](#), and [tut\\_ex14.c](#).

## 15.8 hpdftbl.h

[Go to the documentation of this file.](#)

```

1
32 #ifndef hpdftbl_H
33 #define hpdftbl_H
34
35 #ifdef __cplusplus
36 // in case we have C++ code, we should use its' types and logic
37 #include <algorithm>
38 typedef std::_Bool _Bool;
39 #endif
40
41 #ifdef __cplusplus
42 extern "C" {
43 #endif
44
45 #ifndef TRUE
46 #define TRUE 1
47 #endif
48
49 #ifndef FALSE
50 #define FALSE 0
51 #endif
52
53 #ifndef max
54 #define max(a,b) (((a)>(b)) ? (a):(b))
55 #define min(a,b) (((a)<(b)) ? (a):(b))
56 #endif
57
58 extern int hpdftbl_err_code ;
59
60 extern int hpdftbl_err_row ;
61
62 extern int hpdftbl_err_col ;
63
64 #define HPDF_FF_TIMES "Times-Roman"
65 #define HPDF_FF_TIMES_ITALIC "Times-Italic"
66 #define HPDF_FF_TIMES_BOLD "Times-Bold"
67 #define HPDF_FF_TIMES_BOLDITALIC "Times-BoldItalic"
68 #define HPDF_FF_HELVETICA "Helvetica"
69 #define HPDF_FF_HELVETICA_ITALIC "Helvetica-Oblique"
70 #define HPDF_FF_HELVETICA_BOLD "Helvetica-Bold"
71 #define HPDF_FF_HELVETICA_BOLDITALIC "Helvetica-BoldOblique"
72 #define HPDF_FF_COURIER "Courier"
73 #define HPDF_FF_COURIER_BOLD "Courier-Bold"
74 #define HPDF_FF_COURIER_ITALIC "Courier-Oblique"
75 #define HPDF_FF_COURIER_BOLDITALIC "Courier-BoldOblique"
76
77 #ifdef __cplusplus
78 #define _TO_HPDEF_RGB(r, g, b) \
79 { r / 255.0f, g / 255.0f, b / 255.0f }
80 #else
81 #define _TO_HPDEF_RGB(r, g, b) \
82 (HPDEF_RGBColor) { r / 255.0f, g / 255.0f, b / 255.0f }
83 #endif
84
85 #ifdef __cplusplus
86 #define HPDEF_COLOR_DARK_RED { 0.6f, 0.0f, 0.0f }
87 #define HPDEF_COLOR_RED { 1.0f, 0.0f, 0.0f }
88 #define HPDEF_COLOR_LIGHT_GREEN { 0.9f, 1.0f, 0.9f }
89 #define HPDEF_COLOR_GREEN { 0.4f, 0.9f, 0.4f }
90 #define HPDEF_COLOR_DARK_GREEN { 0.05f, 0.37f, 0.02f }
91 #define HPDEF_COLOR_DARK_GRAY { 0.2f, 0.2f, 0.2f }
92 #define HPDEF_COLOR_LIGHT_GRAY { 0.9f, 0.9f, 0.9f }
93 #define HPDEF_COLOR_XLIGHT_GRAY { 0.95f, 0.95f, 0.95f }
94 #define HPDEF_COLOR_GRAY { 0.5f, 0.5f, 0.5f }
95 #define HPDEF_COLOR_SILVER { 0.75f, 0.75f, 0.75f }
96 #define HPDEF_COLOR_LIGHT_BLUE { 1.0f, 1.0f, 0.9f }
97 #define HPDEF_COLOR_BLUE { 0.0f, 0.0f, 1.0f }
98 #define HPDEF_COLOR_DARK_BLUE { 0.0f, 0.0f, 0.6f }
99 #define HPDEF_COLOR_WHITE { 1.0f, 1.0f, 1.0f }
100 #define HPDEF_COLOR_BLACK { 0.0f, 0.0f, 0.0f }

```

```

116
117 #else
118
119 #define HPDF_COLOR_DARK_RED      (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
120 #define HPDF_COLOR_RED          (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
121 #define HPDF_COLOR_LIGHT_GREEN  (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
122 #define HPDF_COLOR_GREEN        (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
123 #define HPDF_COLOR_DARK_GREEN   (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
124 #define HPDF_COLOR_DARK_GRAY    (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
125 #define HPDF_COLOR_LIGHT_GRAY   (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
126 #define HPDF_COLOR_XLIGHT_GRAY  (HPDF_RGBColor) { 0.95f, 0.95f, 0.95f }
127 #define HPDF_COLOR_GRAY         (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
128 #define HPDF_COLOR_SILVER       (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
129 #define HPDF_COLOR_LIGHT_BLUE   (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
130 #define HPDF_COLOR_BLUE         (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
131 #define HPDF_COLOR_DARK_BLUE    (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
132 #define HPDF_COLOR_WHITE        (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
133 #define HPDF_COLOR_BLACK        (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
134
135 #endif
136
137 #define HPDF_COLOR_ORANGE        _TO_HPDLF_RGB(0xF5, 0xD0, 0x98);
138 #define HPDF_COLOR_ALMOST_BLACK _TO_HPDLF_RGB(0x14, 0x14, 0x14);
139
140 #define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5
141
142
143
144 #define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"
145
146 #define HPDFTBL_DEFAULT_SOURCE_ENCODING "UTF-8"
147
148
149 #define A4PAGE_HEIGHT_CM 29.7
150
151 #define A4PAGE_WIDTH_CM 21.0
152
153 #define A3PAGE_HEIGHT_CM 42.0
154
155 #define A3PAGE_WIDTH_CM 29.7
156
157 #define LETTERPAGE_HEIGHT_CM 27.9
158
159 #define LETTERPAGE_WIDTH_CM 21.6
160
161 #define LEGALPAGE_HEIGHT_CM 35.6
162
163 #define LEGALPAGE_WIDTH_CM 21.6
164
165 #define HPDFTBL_END_CELLSPECS {0, 0, 0, 0, 0, 0, 0, 0, 0}
166
167 #define HPDF_COLOR_FROMRGB(r, g, b) (HPDF_RGBColor){(r)/255.0, (g)/255.0, (b)/255.0}
168
169 #define HPDFTBL_MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0
170
171 #define hpdfdbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54*72)
172
173 #define _HPDFTBL_SET_ERR(t, err, r, c) do {hpdfdbl_err_code=err;hpdfdbl_err_row=r;hpdfdbl_err_col=c;
174     if(hpdfdbl_err_handler){hpdfdbl_err_handler(t,r,c,err);} } while(0)
175
176 #define _HPDFTBL_CHK_TABLE(t) do {if(NULL == t)
177     {hpdfdbl_err_code=-3;hpdfdbl_err_row=-1;hpdfdbl_err_col=-1;return -1;} } while(0)
178
179 #define _HPDFTBL_IDX(r, c) (r*t->cols+c)
180
181
182 typedef enum hpdfdbl_text_align {
183     LEFT = 0,
184     CENTER = 1,
185     RIGHT = 2
186 } hpdfdbl_text_align_t;
187
188 typedef struct text_style {
189     char *font;
190     HPDF_REAL fsize;
191     HPDF_RGBColor color;
192     HPDF_RGBColor background;
193     hpdfdbl_text_align_t halign;
194 } hpdf_text_style_t;
195
196 typedef char *(*hpdfdbl_content_callback_t)(void *, size_t, size_t);
197
198 typedef void (*hpdfdbl_canvas_callback_t)(HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_REAL,
199     HPDF_REAL, HPDF_REAL,
200     HPDF_REAL);
201
202 typedef _Bool (*hpdfdbl_content_style_callback_t)(void *, size_t, size_t, char *content,
203     hpdf_text_style_t *);
204

```



```

311 typedef enum hpdfctl_dashstyle {
312     LINE_SOLID = 0,
313     LINE_DOT1 = 1,
314     LINE_DOT2 = 2,
315     LINE_DOT3 = 3,
316     LINE_DASH1 = 4,
317     LINE_DASH2 = 5,
318     LINE_DASH3 = 6,
319     LINE_DASH4 = 7,
320     LINE_DASHDOT1 = 8,
321     LINE_DASHDOT2 = 9
322 } hpdfctl_line_dashstyle_t;
323
324 typedef struct grid_style {
325     HPDF_REAL width;
326     HPDF_RGBColor color;
327     hpdfctl_line_dashstyle_t line_dashstyle;
328 } hpdfctl_grid_style_t;
329
330 struct hpdfctl_cell {
331     char *label;
332     char *content;
333     size_t colspan;
334     size_t rowspan;
335     HPDF_REAL height;
336     HPDF_REAL width;
337     HPDF_REAL delta_x;
338     HPDF_REAL delta_y;
339     HPDF_REAL textwidth;
340     hpdfctl_content_callback_t content_cb;
341     hpdfctl_content_callback_t label_cb;
342     hpdfctl_content_style_callback_t style_cb;
343     hpdfctl_canvas_callback_t canvas_cb;
344     hpdf_text_style_t content_style;
345     struct hpdfctl_cell *parent_cell;
346 };
347
348 typedef struct hpdfctl_cell hpdfctl_cell_t;
349
350 struct hpdfctl {
351     HPDF_Doc pdf_doc;
352     HPDF_Page pdf_page;
353     size_t cols;
354     size_t rows;
355     HPDF_REAL posx;
356     HPDF_REAL posy;
357     HPDF_REAL height;
358     HPDF_REAL minheight;
359     HPDF_REAL bottom_vmargin_factor;
360     HPDF_REAL width;
361     void *tag;
362     char *title_txt;
363     hpdf_text_style_t title_style;
364     hpdf_text_style_t header_style;
365     _Bool use_header_row;
366     hpdf_text_style_t label_style;
367     _Bool use_cell_labels;
368     _Bool use_label_grid_style;
369     hpdfctl_content_callback_t label_cb;
370     hpdf_text_style_t content_style;
371     hpdfctl_content_callback_t content_cb;
372     hpdfctl_content_style_callback_t content_style_cb;
373     hpdfctl_canvas_callback_t canvas_cb;
374     hpdfctl_cell_t *cells;
375     hpdfctl_grid_style_t outer_grid;
376     hpdfctl_grid_style_t inner_vgrid;
377     hpdfctl_grid_style_t inner_hgrid;
378     hpdfctl_grid_style_t inner_tgrid;
379     _Bool use_zebra;
380     int zebra_phase;
381     HPDF_RGBColor zebra_color1;
382     HPDF_RGBColor zebra_color2;
383     float *col_width_percent;
384 };
385
386 typedef struct hpdfctl *hpdfctl_t;
387
388 typedef void (*hpdfctl_callback_t) (hpdfctl_t);
389
390 typedef struct hpdfctl_cell_spec {
391     size_t row;
392     size_t col;
393     unsigned rowspan;
394     unsigned colspan;
395     char *label;
396     hpdfctl_content_callback_t content_cb;
397     hpdfctl_content_callback_t label_cb;

```

```

511     hpdftbl_content_style_callback_t style_cb;
513     hpdftbl_canvas_callback_t canvas_cb;
514 } hpdftbl_cell_spec_t;
515
516 typedef struct hpdftbl_spec {
517     char *title;
518     _Bool use_header;
519     _Bool use_labels;
520     _Bool use_labelgrid;
521     size_t rows;
522     size_t cols;
523     HPDF_REAL xpos;
524     HPDF_REAL ypos;
525     HPDF_REAL width;
526     HPDF_REAL height;
527     hpdftbl_content_callback_t content_cb;
528     hpdftbl_content_callback_t label_cb;
529     hpdftbl_content_style_callback_t style_cb;
530     hpdftbl_callback_t post_cb;
531     hpdftbl_cell_spec_t *cell_spec;
532 } hpdftbl_spec_t;
533
534 typedef struct hpdftbl_theme {
535     hpdf_text_style_t content_style;
536     hpdf_text_style_t label_style;
537     hpdf_text_style_t header_style;
538     hpdf_text_style_t title_style;
539     hpdftbl_grid_style_t outer_border;
540     _Bool use_labels;
541     _Bool use_label_grid_style;
542     _Bool use_header_row;
543     hpdftbl_grid_style_t inner_vborder;
544     hpdftbl_grid_style_t inner_hborder;
545     hpdftbl_grid_style_t inner_tborder;
546     _Bool use_zebra;
547     int zebra_phase;
548     HPDF_RGBColor zebra_color1;
549     HPDF_RGBColor zebra_color2;
550     HPDF_REAL bottom_vmargin_factor;
551 } hpdftbl_theme_t;
552
553 typedef void (*hpdftbl_error_handler_t)(hpdftbl_t, int, int, int);
554
555 static hpdftbl_error_handler_t hpdftbl_err_handler = NULL;
556
557 /*
558  * Table creation and destruction function
559  */
560 hpdftbl_t
561 hpdftbl_create(size_t rows, size_t cols);
562
563 hpdftbl_t
564 hpdftbl_create_title(size_t rows, size_t cols, char *title);
565
566 int
567 hpdftbl_stroke(HPDF_Doc pdf,
568                HPDF_Page page, hpdftbl_t t,
569                HPDF_REAL xpos, HPDF_REAL ypos,
570                HPDF_REAL width, HPDF_REAL height);
571
572 int
573 hpdftbl_stroke_from_data(HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t *tbl_spec, hpdftbl_theme_t
574                          *theme);
575
576 int
577 hpdftbl_destroy(hpdftbl_t t);
578
579 int
580 hpdftbl_get_last_auto_height(HPDF_REAL *height);
581
582 void
583 hpdftbl_set_anchor_top_left(_Bool anchor);
584
585 _Bool
586 hpdftbl_get_anchor_top_left(void);
587
588 /*
589  * Table error handling functions
590  */
591 hpdftbl_error_handler_t
592 hpdftbl_set_errhandler(hpdftbl_error_handler_t);
593
594 const char *
595 hpdftbl_get_errstr(int err);
596
597 const char *
598 hpdftbl_hpdf_get_errstr(HPDF_STATUS err_code);

```

```

651
652 int
653 hpdfctl_get_last_errcode(const char **errstr, int *row, int *col);
654
655 void
656 hpdfctl_default_table_error_handler(hpdfctl_t t, int r, int c, int err);
657
658 /*
659  * Theme handling functions
660 */
661 int
662 hpdfctl_apply_theme(hpdfctl_t t, hpdfctl_theme_t *theme);
663
664 hpdfctl_theme_t *
665 hpdfctl_get_default_theme(void);
666
667 int
668 hpdfctl_destroy_theme(hpdfctl_theme_t *theme);
669
670 /*
671  * Table layout adjusting functions
672 */
673
674 void
675 hpdfctl_set_bottom_vmargin_factor(hpdfctl_t t, HPDF_REAL f);
676
677 int
678 hpdfctl_set_min_rowheight(hpdfctl_t t, float h);
679
680 int
681 hpdfctl_set_colwidth_percent(hpdfctl_t t, size_t c, float w);
682
683 int
684 hpdfctl_clear_spanning(hpdfctl_t t);
685
686 int
687 hpdfctl_set_cellspan(hpdfctl_t t, size_t r, size_t c, size_t rowspan, size_t colspan);
688
689 /*
690  * Table style handling functions
691 */
692 int
693 hpdfctl_set_zebra(hpdfctl_t t, _Bool use, int phase);
694
695 int
696 hpdfctl_set_zebra_color(hpdfctl_t t, HPDF_RGBColor z1, HPDF_RGBColor z2);
697
698 int
699 hpdfctl_use_labels(hpdfctl_t t, _Bool use);
700
701 int
702 hpdfctl_use_labelgrid(hpdfctl_t t, _Bool use);
703
704 int
705 hpdfctl_set_background(hpdfctl_t t, HPDF_RGBColor background);
706
707 int
708 hpdfctl_set_inner_tgrid_style(hpdfctl_t t, HPDF_REAL width, HPDF_RGBColor color,
709                               hpdfctl_line_dashstyle_t dashstyle);
710
711 int
712 hpdfctl_set_inner_vgrid_style(hpdfctl_t t, HPDF_REAL width, HPDF_RGBColor color,
713                               hpdfctl_line_dashstyle_t dashstyle);
714
715 int
716 hpdfctl_set_inner_hgrid_style(hpdfctl_t t, HPDF_REAL width, HPDF_RGBColor color,
717                               hpdfctl_line_dashstyle_t dashstyle);
718
719 int
720 hpdfctl_set_inner_grid_style(hpdfctl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdfctl_line_dashstyle_t
721                               dashstyle);
722
723 int
724 hpdfctl_set_outer_grid_style(hpdfctl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdfctl_line_dashstyle_t
725                               dashstyle);
726
727 int
728 hpdfctl_set_header_style(hpdfctl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
729                               background);
730
731 int

```

```

732 hpdfdbl_set_label_style(hpdfdbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
    background);
733
734 int
735 hpdfdbl_set_row_content_style(hpdfdbl_t t, size_t r, char *font, HPDF_REAL fsize, HPDF_RGBColor color,
736     HPDF_RGBColor background);
737
738 int
739 hpdfdbl_set_col_content_style(hpdfdbl_t t, size_t c, char *font, HPDF_REAL fsize, HPDF_RGBColor color,
740     HPDF_RGBColor background);
741
742 int
743 hpdfdbl_set_content_style(hpdfdbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
    background);
744
745 int
746 hpdfdbl_set_cell_content_style(hpdfdbl_t t, size_t r, size_t c, char *font, HPDF_REAL fsize,
    HPDF_RGBColor color,
747     HPDF_RGBColor background);
748
749 int
750 hpdfdbl_set_title_style(hpdfdbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
    background);
751
752 /*
753  * Table content handling
754  */
755 int
756 hpdfdbl_set_cell(hpdfdbl_t t, int r, int c, char *label, char *content);
757
758 int
759 hpdfdbl_set_tag(hpdfdbl_t t, void *tag);
760
761 int
762 hpdfdbl_set_title(hpdfdbl_t t, char *title);
763
764 int
765 hpdfdbl_set_title_halign(hpdfdbl_t t, hpdfdbl_text_align_t align);
766
767 int
768 hpdfdbl_set_labels(hpdfdbl_t t, char **labels);
769
770 int
771 hpdfdbl_set_content(hpdfdbl_t t, char **content);
772
773 /*
774  * Table callback functions
775  */
776 int
777 hpdfdbl_set_content_cb(hpdfdbl_t t, hpdfdbl_content_callback_t cb);
778
779 int
780 hpdfdbl_set_cell_content_cb(hpdfdbl_t t, size_t r, size_t c, hpdfdbl_content_callback_t cb);
781
782 int
783 hpdfdbl_set_cell_content_style_cb(hpdfdbl_t t, size_t r, size_t c, hpdfdbl_content_style_callback_t cb);
784
785 int
786 hpdfdbl_set_content_style_cb(hpdfdbl_t t, hpdfdbl_content_style_callback_t cb);
787
788 int
789 hpdfdbl_set_label_cb(hpdfdbl_t t, hpdfdbl_content_callback_t cb);
790
791 int
792 hpdfdbl_set_cell_label_cb(hpdfdbl_t t, size_t r, size_t c, hpdfdbl_content_callback_t cb);
793
794 int
795 hpdfdbl_set_canvas_cb(hpdfdbl_t t, hpdfdbl_canvas_callback_t cb);
796
797 int
798 hpdfdbl_set_cell_canvas_cb(hpdfdbl_t t, size_t r, size_t c, hpdfdbl_canvas_callback_t cb);
799
800 /*
801  * Text encoding
802  */
803 void
804 hpdfdbl_set_text_encoding(char *target, char *source);
805
806 int
807 hpdfdbl_encoding_text_out(HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, char *text);
808
809 /*
810  * Misc utility and widget functions
811  */
812
813 void
814 HPDF_RoundedCornerRectangle(HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL

```

```

    height,
815                                     HPDF_REAL rad);
816
817 void
818 hpdftbl_stroke_grid(HPDF_Doc pdf, HPDF_Page page);
819
820 void
821 hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
822                                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
823                                     HPDF_RGBColor on_color, HPDF_RGBColor off_color,
824                                     HPDF_RGBColor on_background, HPDF_RGBColor off_background,
825                                     HPDF_REAL fsize,
826                                     const char *letters, _Bool *state);
827
828 void
829 hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
830                             HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, _Bool
831                             state);
832
833 void
834 hpdftbl_widget_hbar(HPDF_Doc doc, HPDF_Page page,
835                    HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
836                    HPDF_RGBColor color, float val, _Bool hide_val);
837
838 void
839 hpdftbl_widget_segment_hbar(HPDF_Doc doc, HPDF_Page page,
840                             HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
841                             size_t num_segments, HPDF_RGBColor on_color, double val_percent,
842                             _Bool hide_val);
843
844 void
845 hpdftbl_widget_strength_meter(HPDF_Doc doc, HPDF_Page page,
846                              HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
847                              size_t num_segments, HPDF_RGBColor on_color, size_t num_on_segments);
848 #ifdef    __cplusplus
849 }
850 #endif
851
852 #endif    /* hpdftbl_H */

```

## 15.9 hpdftbl\_errstr.c File Reference

Utility module to translate HPDF error codes to human readable strings.

```
#include <hpdf.h>
```

### Data Structures

- struct [hpdftbl\\_errcode\\_entry](#)  
*An entry in the error string table.*

### Functions

- const char \* [hpdftbl\\_hpdf\\_get\\_errstr](#) (const HPDF\_STATUS err\_code)  
*Function to return a human readable error string for an error code from Core HPDF library.*

#### 15.9.1 Detailed Description

Utility module to translate HPDF error codes to human readable strings.

## 15.9.2 Function Documentation

### 15.9.2.1 `hpdf_tbl_hpdf_get_errstr()`

```
const char * hpdf_tbl_hpdf_get_errstr (
    const HPDF_STATUS err_code )
```

Function to return a human readable error string for an error code from Core HPDF library.

The various error codes given by the HPDF library can be translated back to a string by the usage of this function. The function will return a pointer to a static string that can not be manipulated.

#### Parameters

<code><i>err_code</i></code>	The error code
------------------------------	----------------

#### Returns

A pointer to an error string, NULL if the error code is invalid

#### See also

[hpdf\\_tbl\\_get\\_errstr\(\)](#)

## 15.10 `hpdf_tbl_grid.c` File Reference

Create a grid on a document for positioning.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hpdf.h>
```

### Functions

- void [hpdf\\_tbl\\_stroke\\_grid](#) (HPDF\_Doc pdf, HPDF\_Page page)

### 15.10.1 Detailed Description

Create a grid on a document for positioning.

### 15.10.2 Function Documentation

### 15.10.2.1 hpdfdbl\_stroke\_grid()

```
void hpdfdbl_stroke_grid (
    HPDF_Doc pdf,
    HPDF_Page page )
```

Stroke a point grid on specified page to make it easier to position text and tables.

#### Parameters

<i>pdf</i>	Document handle
<i>page</i>	Page handle

## 15.11 hpdfdbl\_theme.c File Reference

Functions for theme handling.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <hpdf.h>
#include "hpdfdbl.h"
```

### Macros

- #define **HPDFTBL\_DEFAULT\_TITLE\_STYLE** (*hpdf\_text\_style\_t*){HPDF\_FF\_HELVETICA\_BOLD,11,(HPDF\_↵\_RGBColor){0,0,0},{HPDF\_RGBColor){0.9f,0.9f,0.9f}, **LEFT**}  
*Default style for table title.*
- #define **HPDFTBL\_DEFAULT\_HEADER\_STYLE** (*hpdf\_text\_style\_t*){HPDF\_FF\_HELVETICA\_BOLD,10,(HPDF\_↵\_RGBColor){0,0,0},{HPDF\_RGBColor){0.9f,0.9f,0.9f}, **CENTER**}  
*Default style for table header row.*
- #define **HPDFTBL\_DEFAULT\_LABEL\_STYLE** (*hpdf\_text\_style\_t*){HPDF\_FF\_TIMES\_ITALIC,9,(HPDF\_↵\_RGBColor){0.4f,0.4f,0.4f},{HPDF\_RGBColor){1,1,1}, **LEFT**}  
*Default style for table header row.*
- #define **HPDFTBL\_DEFAULT\_CONTENT\_STYLE** (*hpdf\_text\_style\_t*){HPDF\_FF\_COURIER,10,(HPDF\_↵\_RGBColor){0.2f,0.2f,0.2f},{HPDF\_RGBColor){1,1,1}, **LEFT**}  
*Default style for table header row.*
- #define **HPDFTBL\_DEFAULT\_INNER\_VGRID\_STYLE** (*hpdfdbl\_grid\_style\_t*){0.7, (HPDF\_RGBColor){0.↵5f,0.5f,0.5f},0}  
*Default style for table vertical inner grid.*
- #define **HPDFTBL\_DEFAULT\_INNER\_HGRID\_STYLE** (*hpdfdbl\_grid\_style\_t*){0.7, (HPDF\_RGBColor){0.↵5f,0.5f,0.5f},0}  
*Default style for table horizontal inner grid.*
- #define **HPDFTBL\_DEFAULT\_OUTER\_GRID\_STYLE** (*hpdfdbl\_grid\_style\_t*){1.0f, (HPDF\_RGBColor){0.↵2f,0.2f,0.2f},0}  
*Default style for table outer grid (border)*
- #define **HPDFTBL\_DEFAULT\_ZEBRA\_COLOR1** HPDF\_COLOR\_WHITE  
*Default style for alternating row backgrounds color 1.*
- #define **HPDFTBL\_DEFAULT\_ZEBRA\_COLOR2** HPDF\_COLOR\_XLIGHT\_GRAY  
*Default style for alternating row backgrounds color 2.*

## Functions

- int `hpdftbl_apply_theme` (`hpdftbl_t` t, `hpdftbl_theme_t` \*theme)  
*Apply a specified theme to a table.*
- `hpdftbl_theme_t` \* `hpdftbl_get_default_theme` (void)  
*Return the default theme.*
- int `hpdftbl_destroy_theme` (`hpdftbl_theme_t` \*theme)  
*Destroy existing theme structure and free memory.*

### 15.11.1 Detailed Description

Functions for theme handling.

#### Author

Johan Persson ( [johan162@gmail.com](mailto:johan162@gmail.com) )

Copyright (C) 2022 Johan Persson

#### See also

LICENSE

Released under the MIT License

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.

### 15.11.2 Macro Definition Documentation

#### 15.11.2.1 HPDFTBL\_DEFAULT\_CONTENT\_STYLE

```
#define HPDFTBL_DEFAULT_CONTENT_STYLE (hpdf_text_style_t) {HPDF_FF_COURIER,10, (HPDF_RGBColor) {0.↵
2f,0.2f,0.2f}, (HPDF_RGBColor) {1,1,1}, LEFT}
```

Default style for table header row.

#### See also

[hpdftbl\\_set\\_content\\_style\(\)](#)



### 15.11.2.2 HPDFTBL\_DEFAULT\_HEADER\_STYLE

```
#define HPDFTBL_DEFAULT_HEADER_STYLE (hpdf_text_style_t){HPDF_FF_HELVETICA_BOLD,10,(HPDF_↵  
RGBColor){0,0,0},(HPDF_RGBColor){0.9f,0.9f,0.97f}, CENTER}
```

Default style for table header row.

See also

[hpdftbl\\_set\\_header\\_style\(\)](#)

### 15.11.2.3 HPDFTBL\_DEFAULT\_INNER\_HGRID\_STYLE

```
#define HPDFTBL_DEFAULT_INNER_HGRID_STYLE (hpdftbl_grid_style_t){0.7, (HPDF_RGBColor){0.↵  
5f,0.5f,0.5f},0}
```

Default style for table horizontal inner grid.

See also

[hpdftbl\\_set\\_inner\\_hgrid\\_style\(\)](#)

### 15.11.2.4 HPDFTBL\_DEFAULT\_INNER\_VGRID\_STYLE

```
#define HPDFTBL_DEFAULT_INNER_VGRID_STYLE (hpdftbl_grid_style_t){0.7, (HPDF_RGBColor){0.↵  
5f,0.5f,0.5f},0}
```

Default style for table vertical inner grid.

See also

[hpdftbl\\_set\\_inner\\_vgrid\\_style\(\)](#)

### 15.11.2.5 HPDFTBL\_DEFAULT\_LABEL\_STYLE

```
#define HPDFTBL_DEFAULT_LABEL_STYLE (hpdf_text_style_t){HPDF_FF_TIMES_ITALIC,9,(HPDF_RGBColor){0.↵  
4f,0.4f,0.4f},(HPDF_RGBColor){1,1,1}, LEFT}
```

Default style for table header row.

See also

[hpdftbl\\_set\\_label\\_style\(\)](#)

### 15.11.2.6 HPDFTBL\_DEFAULT\_OUTER\_GRID\_STYLE

```
#define HPDFTBL_DEFAULT_OUTER_GRID_STYLE (hpdftbl_grid_style_t){1.0f, (HPDF_RGBColor){0.↵  
2f, 0.2f, 0.2f}, 0}
```

Default style for table outer grid (border)

See also

[hpdftbl\\_set\\_outer\\_grid\\_style\(\)](#)

### 15.11.2.7 HPDFTBL\_DEFAULT\_ZEBRA\_COLOR1

```
#define HPDFTBL_DEFAULT_ZEBRA_COLOR1 HPDF_COLOR_WHITE
```

Default style for alternating row backgrounds color 1.

**Todo** Implement zebra table coloring

### 15.11.2.8 HPDFTBL\_DEFAULT\_ZEBRA\_COLOR2

```
#define HPDFTBL_DEFAULT_ZEBRA_COLOR2 HPDF_COLOR_XLIGHT_GRAY
```

Default style for alternating row backgrounds color 2.

**Todo** Implement zebra table coloring

## 15.11.3 Function Documentation

### 15.11.3.1 hpdftbl\_apply\_theme()

```
int hpdftbl_apply_theme (  
    hpdftbl_t t,  
    hpdftbl_theme_t * theme )
```

Apply a specified theme to a table.

The default table theme can be retrieved with [hpdftbl\\_get\\_default\\_theme\(\)](#)

## Parameters

<i>t</i>	Table handle
<i>theme</i>	Theme reference

## Returns

0 on success, -1 on failure

## See also

[hpdftbl\\_get\\_default\\_theme\(\)](#)

Referenced by [hpdftbl\\_create\\_title\(\)](#), and [hpdftbl\\_stroke\\_from\\_data\(\)](#).

### 15.11.3.2 hpdftbl\_destroy\_theme()

```
int hpdftbl_destroy_theme (
    hpdftbl_theme_t * theme )
```

Destroy existing theme structure and free memory.

Free all memory allocated by a theme

## Parameters

<i>theme</i>	The theme to free
--------------	-------------------

## Returns

-1 for error , 0 for success

Referenced by [hpdftbl\\_create\\_title\(\)](#).

### 15.11.3.3 hpdftbl\_get\_default\_theme()

```
hpdftbl_theme_t * hpdftbl_get_default_theme (
    void )
```

Return the default theme.

Create and return a theme corresponding to the default table theme. It is the calling functions responsibility to call [hpdftbl\\_destroy\\_theme\(\)](#) to free the allocated memory. The default theme is a good starting point to just make minor modifications without having to define all elements.

**Returns**

A new theme initialized to the default settings

**See also**

[hpdfdbl\\_apply\\_theme\(\)](#)

Referenced by [hpdfdbl\\_create\\_title\(\)](#).

## 15.12 hpdfdbl\_widget.c File Reference

Support for drawing widgets.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <hpdf.h>
#include <string.h>
#include <math.h>
#include "hpdfdbl.h"
```

**Macros**

- `#define TRUE 1`
- `#define FALSE 0`

**Functions**

- void [hpdfdbl\\_table\\_widget\\_letter\\_buttons](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, const HPDF\_RGBColor on\_color, const HPDF\_RGBColor off\_color, const HPDF\_RGBColor on\_background, const HPDF\_RGBColor off\_background, const HPDF\_REAL fsize, const char \*letters, \_Bool \*state)  
*Display an array of letters as a table where each letter is its own "mini" cell and surrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different font and fac colors.*
- void [hpdfdbl\\_widget\\_slide\\_button](#) (HPDF\_Doc doc, HPDF\_Page page, HPDF\_REAL xpos, HPDF\_REAL ypos, HPDF\_REAL width, HPDF\_REAL height, \_Bool state)  
*Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.*
- void [hpdfdbl\\_widget\\_hbar](#) (const HPDF\_Doc doc, const HPDF\_Page page, const HPDF\_REAL xpos, const HPDF\_REAL ypos, const HPDF\_REAL width, const HPDF\_REAL height, const HPDF\_RGBColor color, const float val, const \_Bool hide\_val)  
*Draw a horizontal partially filled bar to indicate an analog (percentage) value.*
- void [hpdfdbl\\_widget\\_segment\\_hbar](#) (const HPDF\_Doc doc, const HPDF\_Page page, const HPDF\_REAL xpos, const HPDF\_REAL ypos, const HPDF\_REAL width, const HPDF\_REAL height, const size\_t num\_segments, const HPDF\_RGBColor on\_color, const double val\_percent, const \_Bool hide\_val)  
*Draw a horizontal segment meter that can be used to visualize a discrete value.*
- void [hpdfdbl\\_widget\\_strength\\_meter](#) (const HPDF\_Doc doc, const HPDF\_Page page, const HPDF\_REAL xpos, const HPDF\_REAL ypos, const HPDF\_REAL width, const HPDF\_REAL height, const size\_t num\_segments, const HPDF\_RGBColor on\_color, const size\_t num\_on\_segments)  
*Draw a phone strength meter.*

## 15.12.1 Detailed Description

Support for drawing widgets.

## 15.12.2 Macro Definition Documentation

### 15.12.2.1 FALSE

```
#define FALSE 0
```

C Boolean false value

### 15.12.2.2 TRUE

```
#define TRUE 1
```

C Boolean truth value

## 15.12.3 Function Documentation

### 15.12.3.1 hpdftbl\_table\_widget\_letter\_buttons()

```
void hpdftbl_table_widget_letter_buttons (
    HPDF_Doc doc,
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    HPDF_REAL width,
    HPDF_REAL height,
    const HPDF_RGBColor on_color,
    const HPDF_RGBColor off_color,
    const HPDF_RGBColor on_background,
    const HPDF_RGBColor off_background,
    const HPDF_REAL fsize,
    const char * letters,
    _Bool * state )
```

Display an array of letters as a table where each letter is its own "mini" cell and surrounded by a frame. Each boxed letter can be in an "on" state or "off" state which is illustrated with different font and fac colors.

#### Parameters

<i>doc</i>	HPDF document handle
<i>page</i>	HPDF page handle

## Parameters

<i>xpos</i>	X-öosition of cell
<i>ypos</i>	Y-Position of cell
<i>width</i>	Width of cell
<i>height</i>	Height of cell
<i>on_color</i>	The font color in "on" state
<i>off_color</i>	The font color in "off" state
<i>on_background</i>	The face color in "on" state
<i>off_background</i>	The face color in "off" state
<i>fsize</i>	The font size
<i>letters</i>	What letters to have in the boxes
<i>state</i>	What state each boxed letter should be (0=off, 1=on)

15.12.3.2 `hpdf_tbl_widget_hbar()`

```
void hpdf_tbl_widget_hbar (
    const HPDF_Doc doc,
    const HPDF_Page page,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    const HPDF_REAL height,
    const HPDF_RGBColor color,
    const float val,
    const _Bool hide_val )
```

Draw a horizontal partially filled bar to indicate an analog (percentage) value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

## Parameters

<i>doc</i>	HPDF Document handle
<i>page</i>	HPDF Page handle
<i>xpos</i>	Lower left x
<i>ypos</i>	Lower left y
<i>width</i>	Width of meter
<i>height</i>	Height of meter
<i>color</i>	Fill color for bar
<i>val</i>	Percentage fill in range [0.0, 100.0]
<i>hide_val</i>	TRUE to hide the value (in percent) at the right end of the entire bar

### 15.12.3.3 hpdfdbl\_widget\_segment\_hbar()

```
void hpdfdbl_widget_segment_hbar (
    const HPDF_Doc doc,
    const HPDF_Page page,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    const HPDF_REAL height,
    const size_t num_segments,
    const HPDF_RGBColor on_color,
    const double val_percent,
    const _Bool hide_val )
```

Draw a horizontal segment meter that can be used to visualize a discrete value.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

#### Parameters

<i>doc</i>	HPDF Document handle
<i>page</i>	HPDF Page handle
<i>xpos</i>	Lower left x
<i>ypos</i>	Lower left y
<i>width</i>	Width of meter
<i>height</i>	Height of meter
<i>num_segments</i>	Total number of segments
<i>on_color</i>	Color for "on" segment
<i>val_percent</i>	To what extent should the bars be filled (as a value 0.0 - 1.0)
<i>hide_val</i>	TRUE to hide the value (in percent) at the right end of the entire bar

### 15.12.3.4 hpdfdbl\_widget\_slide\_button()

```
void hpdfdbl_widget_slide_button (
    HPDF_Doc doc,
    HPDF_Page page,
    HPDF_REAL xpos,
    HPDF_REAL ypos,
    HPDF_REAL width,
    HPDF_REAL height,
    _Bool state )
```

Table widget that draws a sliding on/off switch. Meant to be used in a canvas callback to display a boolean value.

This function can not be used directly as a canvas callback since it needs the state of the button as an argument. Instead create a simple canvas callback that determines the wanted state and then just passes on all argument to this widget function.

## Parameters

<i>doc</i>	HPDF document handle
<i>page</i>	HPDF page handle
<i>xpos</i>	X-Position of cell
<i>ypos</i>	Y-Position of cell
<i>width</i>	Width of cell
<i>height</i>	Height of cell
<i>state</i>	State of button On/Off

**15.12.3.5 hpdfctl\_widget\_strength\_meter()**

```
void hpdfctl_widget_strength_meter (
    const HPDF_Doc doc,
    const HPDF_Page page,
    const HPDF_REAL xpos,
    const HPDF_REAL ypos,
    const HPDF_REAL width,
    const HPDF_REAL height,
    const size_t num_segments,
    const HPDF_RGBColor on_color,
    const size_t num_on_segments )
```

Draw a phone strength meter.

This function can not be used directly as a canvas callback since it needs additional parameters. Instead create a simple canvas callback that gives the additional parameters.

## Parameters

<i>doc</i>	HPDF Document handle
<i>page</i>	HPDF Page handle
<i>xpos</i>	Lower left x
<i>ypos</i>	Lower left y
<i>width</i>	Width of meter
<i>height</i>	Height of meter
<i>num_segments</i>	Total number of segments
<i>on_color</i>	Color for "on" segment
<i>num_on_segments</i>	Number of on segments



# Chapter 16

## Example Documentation

### 16.1 example01.c

A collection of more and less advanced examples in one file. For learning the library it is better to start with the organized tutorial examples like [tut\\_ex01.c](#) and [tut\\_ex02.c](#)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#include <sys/stat.h>
#include <libgen.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// The output after running the program will be written to this file
#ifdef _WIN32
#define OUTPUT_FILE "example01.pdf"
#else
#define OUTPUT_FILE "/tmp/example01.pdf"
#endif
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
// Global handlers to the HPDF document and page
HPDF_Doc pdf_doc;
HPDF_Page pdf_page;
// We use some dummy data to populate the tables
#define MAX_NUM_ROWS 10
#define MAX_NUM_COLS 10
// Data array with string pointers to dummy data and cell labels
// The actual storage for the strings are dynamically allocated.
char *labels[MAX_NUM_ROWS * MAX_NUM_COLS];
char *content[MAX_NUM_ROWS * MAX_NUM_COLS];
// Create two arrays with dummy data to populate the tables
void
setup_dummy_data(void) {
    char buff[255];
    size_t cnt = 0;
    for (size_t r = 0; r < MAX_NUM_ROWS; r++) {
        for (size_t c = 0; c < MAX_NUM_COLS; c++) {
#ifdef _WIN32 || defined __WIN32__
            sprintf(buff, "Label %i:", cnt);
            labels[cnt] = _strdup(buff);
            sprintf(buff, "Content %i", cnt);
            content[cnt] = _strdup(buff);
#else
            snprintf(buff, sizeof(buff), "Label %zu:", cnt);
            labels[cnt] = strdup(buff);
            snprintf(buff, sizeof(buff), "Contentg %zu", cnt);
#endif
            cnt++;
        }
    }
}
```

```

        content[cnt] = strdup(buff);
#endif
        cnt++;
    }
}
#endifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human-readable string
static void
error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
              void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int) error_no, (int) detail_no);
    longjmp(env, 1);
}
#if !(defined _WIN32 || defined __WIN32__)
// We don't use the page header on Windooze systems
static char *
cb_name(void *tag, size_t r, size_t c) {
    static char buf[256];
    struct utsname sysinfo;
    if (-1 == uname(&sysinfo)) {
        return "???";
    } else {
        snprintf(buf, sizeof(buf), "Name: %s, Kernel: %s %s", sysinfo.nodename,
                sysinfo.sysname, sysinfo.release);
        return buf;
    }
}
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if (! static_date) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
void
cb_draw_segment_hbar(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                    size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                    HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL segment_tot_width = width * 0.5;
    const HPDF_REAL segment_height = height / 3;
    const HPDF_REAL segment_xpos = xpos + 40;
    const HPDF_REAL segment_ypos = ypos + 4;
    const size_t num_segments = 10;
    const HPDF_RGBColor on_color = HPDF_COLOR_GREEN;
    const double val_percent = 0.4;
    const _Bool val_text_hide = FALSE;
    hpdftbl_widget_segment_hbar(
        doc, page, segment_xpos, segment_ypos, segment_tot_width,
        segment_height, num_segments, on_color, val_percent, val_text_hide);
}
void
cb_draw_hbar(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r, size_t c,
            HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width,
            HPDF_REAL height) {
    const HPDF_REAL wwidth = width * 0.5;
    const HPDF_REAL wheight = height / 3;
    const HPDF_REAL wxpos = xpos + 40;
    const HPDF_REAL wypos = ypos + 4;
    const HPDF_RGBColor color = HPDF_COLOR_GREEN;
    const double val = 0.6;
    const _Bool val_text_hide = FALSE;
    hpdftbl_widget_hbar(doc, page, wxpos, wypos, wwidth, wheight, color, val,
                        val_text_hide);
}
void
cb_draw_slider(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r, size_t c,
              HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width,
              HPDF_REAL height) {
    /*
     * void
     hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
                                HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL
height, _Bool state)
     */
    const HPDF_REAL wwidth = 37;
    const HPDF_REAL wheight = 12;

```

```

const HPDF_REAL wxpos = xpos + 70;
const HPDF_REAL wypos = ypos + 5;
// The slide is on for third row and off otherwise
_Bool state = (r == 2);
hpdftbl_widget_slide_button(doc, page, wxpos, wypos, wwidth, wheight,
                             state);
}
void
cb_draw_strength_meter(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                       size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                       HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL wwidth = 35;
    const HPDF_REAL wheight = 20;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 4;
    const size_t num_segments = 5;
    const HPDF_RGBColor on_color = HPDF_COLOR_GREEN;
    const size_t num_on_segments = 3;
    hpdftbl_widget_strength_meter(doc, page, wxpos, wypos, wwidth, wheight,
                                   num_segments, on_color, num_on_segments);
}
void
cb_draw_boxed_letter(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                    size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                    HPDF_REAL width, HPDF_REAL height) {
    /*
     * void
     * hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
     *                                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL
     *                                     height, const HPDF_RGBColor on_color, const HPDF_RGBColor off_color, const
     *                                     HPDF_RGBColor on_background, const HPDF_RGBColor off_background, const HPDF_REAL
     *                                     fsize, const char *letters, _Bool *state )
     */
    const HPDF_REAL wwidth = 60;
    const HPDF_REAL wheight = 15;
    const HPDF_REAL wxpos = xpos + 60;
    const HPDF_REAL wypos = ypos + 4;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_GRAY;
    const HPDF_RGBColor off_color = HPDF_COLOR_GRAY;
    const HPDF_RGBColor on_background = HPDF_COLOR_GREEN;
    const HPDF_RGBColor off_background = HPDF_COLOR_LIGHT_GRAY;
    const HPDF_REAL fsize = 11;
    const char *letters = "ABCD";
    _Bool state[] = {TRUE, FALSE, TRUE, FALSE};
    hpdftbl_table_widget_letter_buttons(doc, page, wxpos, wypos, wwidth, wheight,
                                         on_color, off_color, on_background,
                                         off_background, fsize, letters, state);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
example_page_header(void) {
    // Specified the layout of each row
    // For a cell where we want dynamic content we must make use of a
    // content-callback that will return a pointer to a static buffer whose
    // content will be displayed in the cell.
    hpdftbl_cell_spec_t tbl1_data[] = {
        // row,col,rowspan,colspan,table-string,content-callback
        {0, 0, 1, 4, "Server info:", cb_name, NULL, NULL, NULL},
        {0, 4, 1, 2, "Date:", cb_date, NULL, NULL, NULL},
        {0, 0, 0, 0, NULL, NULL, NULL, NULL, NULL} /* Sentinel to mark end of data */
    };
    // Overall table layout
    hpdftbl_spec_t tbl1 = {
        .title=NULL, .use_header=0,
        .use_labels=1, .use_labelgrid=1,
        .rows=1, .cols=6,
        .xpos=50, .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1),
        .width=500, .height=0,
        .content_cb=0, .label_cb=0, .style_cb=0, .post_cb=0,
        .cell_spec=tbl1_data
    };
    // Show how to set a specified theme to the table. Since we only use the
    // default theme here we could equally well just have set NULL as the last
    // argument to the hpdftbl_stroke_from_data() function since this is the
    // same specifying the default theme.
    hpdftbl_theme_t *theme = hpdftbl_get_default_theme();
    int ret = hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl1, theme);
    // Should always check for any error
    if (-1 == ret) {
        const char *buf;
        int r, c;
        int tbl_err = hpdftbl_get_last_errcode(&buf, &r, &c);
        fprintf(stderr,
                "**** ERROR in creating table from data. ( %d : \"%s\" ) @ "
                "[%d,%d]\n",

```

```

        tbl_err, buf, r, c);
    }
    // Remember to clean up to avoid memory leak
    hpdftbl_destroy_theme(theme);
}
#endif
// Setup a PDF document with one page
static void
add_a4page(void) {
    pdf_page = HPDF_AddPage(pdf_doc);
    HPDF_Page_SetSize(pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
}
void
stroke_pdfdoc(char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
void
ex_tbl1(void) {
    int num_rows = 5;
    int num_cols = 4;
    char *table_title = "Example 1: Basic table with default theme";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, FALSE);
    //hpdftbl_use_labelgrid(t, TRUE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_tbl2(void) {
    int num_rows = 5;
    int num_cols = 4;
    char *table_title = "Example 2: Basic table with adjusted font styles";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                           title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    // Use bold font for content. Use the C99 way to specify constant structure
    // constants
    const HPDF_RGBColor content_text_color = HPDF_COLOR_DARK_GRAY;
    const HPDF_RGBColor content_bg_color = HPDF_COLOR_LIGHT_BLUE;
    hpdftbl_set_content_style(t, HPDF_FF_COURIER_BOLD, 10,
                             content_text_color, content_bg_color);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, TRUE);
    hpdftbl_use_labelgrid(t, TRUE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_tbl3(void) {
    int num_rows = 9;
    int num_cols = 4;
    char *table_title =
        "Example 3: Table cell spannings and full grid and header";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                           title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    // Use specially formatted header row
    hpdftbl_use_header(t, TRUE);
    // Use full grid and not just the short labelgrid

```

```

    hpdftbl_use_labelgrid(t, FALSE);
    // Use bold font for content. Use the C99 way to specify constant structure
    // constants
    const HPDF_RGBColor content_text_color = HPDF_COLOR_DARK_GRAY;
    const HPDF_RGBColor content_bg_color = HPDF_COLOR_WHITE;
    hpdftbl_set_content_style(t, HPDF_FF_COURIER_BOLD, 10,
                             content_text_color, content_bg_color);
    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, TRUE);
    // Spanning for the header row (row==0)
    // Span cell=(0,1) one row and three columns
    hpdftbl_set_cellspan(t, 0, 1, 1, 3);
    // Span cell=(1,1) one row and three columns
    hpdftbl_set_cellspan(t, 1, 1, 1, 3);
    // Span cell=(2,2) one row and two columns
    hpdftbl_set_cellspan(t, 2, 2, 1, 2);
    // Span cell=(4,1) two rows and three columns
    hpdftbl_set_cellspan(t, 4, 1, 2, 3);
    // Span cell=(7,2) two rows and two columns
    hpdftbl_set_cellspan(t, 7, 2, 2, 2);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}

void
ex_ttbl4(void) {
    const size_t num_rows = 5;
    const size_t num_cols = 4;
    char *table_title = "Example 4: Adjusting look and feel of single cell";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                           title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    // Set the top left and bottom right with orange bg_color
    const HPDF_RGBColor content_bg_color = HPDF_COLOR_ORANGE;
    const HPDF_RGBColor content_text_color = HPDF_COLOR_ALMOST_BLACK;
    hpdftbl_set_cell_content_style(t, 0, 0, HPDF_FF_COURIER_BOLD, 10,
                                   content_text_color, content_bg_color);
    hpdftbl_set_cell_content_style(t, 4, 3, HPDF_FF_COURIER_BOLD, 10,
                                   content_text_color, content_bg_color);

    hpdftbl_set_content(t, content);
    hpdftbl_set_labels(t, labels);
    hpdftbl_use_labels(t, TRUE);
    hpdftbl_use_labelgrid(t, TRUE);
    // First column should be 40% of the total width
    hpdftbl_set_colwidth_percent(t, 0, 40);
    // Span cell=(1,0) one row and two columns
    hpdftbl_set_cellspan(t, 1, 0, 1, 2);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(2);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdftbl_cm2dpi(15);
    HPDF_REAL height = 0; // Calculate height automatically
    if (-1 ==
        hpdftbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height)) {
        const char *errstr;
        int row, col;
        hpdftbl_get_last_errcode(&errstr, &row, &col);
        fprintf(stderr, "ERROR: \"%s\"\\n", errstr);
    }
}

void
ex_ttbl5(void) {
    const int num_rows = 6;
    const int num_cols = 4;
    char *table_title = "Example 5: Using widgets in cells";
    hpdftbl_t t = hpdftbl_create_title(num_rows, num_cols, table_title);
    // Use a red title and center the text
    const HPDF_RGBColor title_text_color = HPDF_COLOR_DARK_RED;
    const HPDF_RGBColor title_bg_color = HPDF_COLOR_LIGHT_GRAY;
    hpdftbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                           title_bg_color);
    hpdftbl_set_title_halign(t, CENTER);
    hpdftbl_set_min_rowheight(t, 20);
    // Install callback for the specified cell where the graphical meter will be
    // drawn

```

```

size_t wrow = 0;
size_t wcol = 0;
content[wrow * num_cols + wcol] = NULL;
labels[wrow * num_cols + wcol] = "Horizontal seg bar:";
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_segment_hbar);
wrow += 1;
content[wrow * num_cols + wcol] = NULL;
labels[wrow * num_cols + wcol] = "Horizontal bar:";
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_hbar);
wrow += 1;
content[wrow * num_cols + wcol] = NULL;
labels[wrow * num_cols + wcol] = "Slider on:";
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_slider);
wrow += 1;
content[wrow * num_cols + wcol] = NULL;
labels[wrow * num_cols + wcol] = "Slider off:";
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_slider);
wrow += 1;
content[wrow * num_cols + wcol] = NULL;
labels[wrow * num_cols + wcol] = "Strength meter:";
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_strength_meter);
wrow += 1;
content[wrow * num_cols + wcol] = NULL;
labels[wrow * num_cols + wcol] = "Boxed letters:";
hpdf_tbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_boxed_letter);
hpdf_tbl_set_content(t, content);
hpdf_tbl_set_labels(t, labels);
hpdf_tbl_use_labels(t, TRUE);
hpdf_tbl_use_labelgrid(t, TRUE);
// First column should be 40% of the total width
hpdf_tbl_set_colwidth_percent(t, 0, 40);
// We let the library automatically determine the height of the table based
// on the font and number of rows.
HPDF_REAL xpos = hpdf_tbl_cm2dpi(2);
HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
HPDF_REAL width = hpdf_tbl_cm2dpi(15);
HPDF_REAL height = 0; // Calculate height automatically
if (-1 ==
    hpdf_tbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height)) {
    const char *errstr;
    int row, col;
    hpdf_tbl_get_last_errcode(&errstr, &row, &col);
    fprintf(stderr, "ERROR: \"%s\"\\n", errstr);
}
}
// Type for the pointer to example stroking functions "void fnc(void)"
typedef void (*t_func_tbl_stroke)(void);
// Silent gcc about unused arguments in the main functions
#ifdef _MSC_VER
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    t_func_tbl_stroke examples[] = {ex_tbl1, ex_tbl2, ex_tbl3, ex_tbl4,
                                    ex_tbl5};

    const size_t num_examples = sizeof(examples) / sizeof(t_func_tbl_stroke);
    printf("Stroking %ld examples.\\n", num_examples);
    // Setup fake exception handling
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    // Get some dummy data to fill the table$
    setup_dummy_data();
    // Setup the basic PDF document
    pdf_doc = HPDF_New(error_handler, NULL);
    HPDF_SetCompressionMode(pdf_doc, HPDF_COMP_ALL);
    for (size_t i = 0; i < num_examples; i++) {
        add_a4page();
    }
#ifdef !defined _WIN32 || defined __WIN32__
    example_page_header();
#endif
    (*examples[i])();
}
if ( 2==argc ) {
    struct stat sb;
    if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
        stroke_pdfdoc(argv[1]);
        return EXIT_SUCCESS;
    }
}
stroke_pdfdoc( OUTPUT_FILE);

```

```

    return (EXIT_SUCCESS);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.2 tut\_ex01.c

The very most basic table with API call to set content in each cell.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
    hpdftbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
    hpdftbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
    hpdftbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one age
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
}

```

```

    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex01(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.3 tut\_ex02.c

Basic table with content data specified as an array.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];

```



```

    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}

void
create_table_ex02(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, 2, 2);
    hpdf_tbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdf_tbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}

void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}

#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex02(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.4 tut\_ex02\_1.c

Basic table with content data specified as an array.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <syslog.h>
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifndef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifndef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            if( 0==r )
                snprintf(buff, sizeof(buff), "Header %zu", cnt);
            else
                snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
void
create_table_ex02_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
}

```

```

    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        closelog();
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex02_1(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.5 tut\_ex03.c

First example with API call to set content in each cell with added labels and shortened grid.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex03(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
}

```

```

    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_cell(tbl, 0, 0, "Label 1", "Cell 0x0");
    hpdftbl_set_cell(tbl, 0, 1, "Label 2", "Cell 0x1");
    hpdftbl_set_cell(tbl, 1, 0, "Label 3", "Cell 1x0");
    hpdftbl_set_cell(tbl, 1, 1, "Label 4", "Cell 1x1");
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, FALSE);
    // We have to specify the top left position on the PDF as well as the width.
    // We let the library automatically determine the height of the table based
    // on the font and number of rows.
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one age
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex03(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.6 tut\_ex04.c

Specifying labels as data array.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif

```

```

#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, content_t *labels, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    *labels = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            snprintf(buff, sizeof(buff), "Label %zu", cnt);
            (*labels)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
void
create_table_ex04(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    //char *table_title = "tut_ex01: 2x2 table";
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_data(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);

    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER

```

```
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdp(&pdf_doc, &pdf_page, FALSE);
    create_table_ex04(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
```

## 16.7 tut\_ex05.c

Set content data specified as an array with added labels and shortened grid.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdp.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdpftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdp error handler which also translates the hpdp error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
    void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
        hpdpftbl_hpdp_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, content_t *labels, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    *labels = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
        }
    }
}
```

```

        snprintf(buff, sizeof(buff), "Label %zu", cnt);
        (*labels)[cnt] = strdup(buff);
        cnt++;
    }
}

void
create_table_ex05(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex05: 2x2 table";
    hpdf_tbl_t tbl = hpdf_tbl_create_title(num_rows, num_cols, table_title);
    content_t content, labels;
    setup_dummy_data(&content, &labels, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    hpdf_tbl_set_labels(tbl, labels);

    hpdf_tbl_use_labels(tbl, TRUE);
    hpdf_tbl_use_labelgrid(tbl, TRUE);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

// Setup a new PDF document with one page
void
setup_hpfd(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdf_tbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}

void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}

#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpfd(&pdf_doc, &pdf_page, FALSE);
    create_table_ex05(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.8 tut\_ex06.c

Use content to set content and labels.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! static_date ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    #if (defined _WIN32 || defined __WIN32__)
        snprintf(buf, sizeof buf, "Content %02i x %02i", r, c);
    #else
        snprintf(buf, sizeof buf, "Content %02zu x %02zu", r, c);
    #endif
    return buf;
}
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    #if (defined _WIN32 || defined __WIN32__)
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
        }
    #else
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
        }
    #endif
    return buf;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex06(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex06: 2x2 table with callbacks";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
}

```



```

    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(12);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex06(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.9 tut\_ex07.c

Expand cells over multiple columns and rows.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>

```

```

#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! static_date ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    #if (defined _WIN32 || defined __WIN32__)
        snprintf(buf, sizeof buf, "Content %02ix%02i", r, c);
    #else
        snprintf(buf, sizeof buf, "Content %02zux%02zu", r, c);
    #endif
    return buf;
}
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    #if (defined _WIN32 || defined __WIN32__)
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
        }
    #else
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
        }
    #endif
    return buf;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex07(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    char *table_title = "tut_ex07: 7x5 table with row and colspans";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    hpdftbl_set_cellspan(tbl, 0, 0, 1, 3);
    hpdftbl_set_cellspan(tbl, 2, 2, 3, 3);
    hpdftbl_set_cellspan(tbl, 3, 0, 4, 1);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page

```

```

    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex07(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.10 tut\_ex08.c

Adjust column width and expand cells over multiple columns and rows.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined __WIN32__ || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined __WIN32__ || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"

```

```

// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if (! static_date ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    #if (defined _WIN32 || defined __WIN32__)
        snprintf(buf, sizeof buf, "Content %2ix%2i", r, c);
    #else
        snprintf(buf, sizeof buf, "Content %zux%zu", r, c);
    #endif
    return buf;
}
static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    #if (defined _WIN32 || defined __WIN32__)
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
        }
    #else
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
        }
    #endif
    return buf;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex08(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    char *table_title = "tut_ex08: 4x4 adjusting col width";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    hpdftbl_use_labels(tbl, TRUE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_label_cb(tbl, cb_labels);
    hpdftbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    hpdftbl_set_cellspan(tbl, 0, 0, 1, 3);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(17);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
}

```

```

    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdf_tbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex08(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.11 tut\_ex09.c

Adjusting font style with a callback.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdf_tbl.h"
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"

```

```

#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\\\", [0x%04X : %d]\\n\",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}

_Bool
cb_style(void *tag, size_t r, size_t c, char *content, hpdf_text_style_t *style) {
    // Format the header row/column with a grey background and Helvetica font while the rest of the
    // table uses "Times Roman"
    if( 0==r || 0==c ) { // Headers
        style->font = HPDF_FF_HELVETICA_BOLD;
        style->fontsize = 12;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_LIGHT_GRAY;
        if ( c > 0 )
            style->halign = CENTER;
        else
            style->halign = LEFT;
    } else { // Content
        style->font = HPDF_FF_TIMES;
        style->fontsize = 11;
        style->color = HPDF_COLOR_BLACK;
        style->background = HPDF_COLOR_WHITE;
        style->halign = CENTER;
    }
    return TRUE;
}

static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    if( 0==r && 0==c ) return NULL;
    if( 0==c ) {
#ifdef _WIN32 || defined __WIN32__
        snprintf(buf, sizeof buf, "Extra long Header %2ix%2i", r, c);
#else
        snprintf(buf, sizeof buf, "Extra long Header %zux%zu", r, c);
#endif
    } else if( 0==r ) {
#ifdef _WIN32 || defined __WIN32__
        snprintf(buf, sizeof buf, "Header %2ix%2i", r, c);
#else
        snprintf(buf, sizeof buf, "Header %zux%zu", r, c);
#endif
    } else {
#ifdef _WIN32 || defined __WIN32__
        snprintf(buf, sizeof buf, "Content %2ix%2i", r, c);
#else
        snprintf(buf, sizeof buf, "Content %zux%zu", r, c);
#endif
    }
    return buf;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex09(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_content_cb(tbl, cb_content);
    hpdftbl_set_content_style_cb(tbl, cb_style);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}

void

```

```

stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\\n");
}

#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    setup_hpdp(&pdf_doc, &pdf_page, FALSE);
    create_table_ex09(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.12 tut\_ex10.c

Adjust column widths and add error handler.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdp.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdpftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdp error handler which also translates the hpdp error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\\n",
        hpdpftbl_hpdp_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}

```

```

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}

void
create_table_ex10(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_set_colwidth_percent(tbl, 0, 30);
    hpdftbl_set_colwidth_percent(tbl, 1, 30);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 4);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}

void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\\n");
}

#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex10(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof(fname), "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```



## 16.13 tut\_ex11.c

Table with header row and error handler.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#else
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#endif
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <syslog.h>
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
        hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
void
create_table_ex11(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdftbl_set_errhandler(hpdftbl_default_table_error_handler);
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_colwidth_percent(tbl, 0, 40);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
```

```

stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\\n");
}

#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        closelog();
        return EXIT_FAILURE;
    }
    setup_hpfd(&pdf_doc, &pdf_page, FALSE);
    create_table_ex11(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.14 tut\_ex12.c

Table with header row and error handler.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpfd.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpfdtbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpfd error handler which also translates the hpfd error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no, void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\\n",
        hpfdtbl_hpfd_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;

```

```

void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}

void
create_table_ex12(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 4;
    const size_t num_cols = 4;
    hpdf_tbl_set_errhandler(hpdf_tbl_default_table_error_handler);
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    hpdf_tbl_use_header(tbl, TRUE);
    hpdf_tbl_set_colwidth_percent(tbl, 0, 40);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(A4PAGE_WIDTH_CM - 5);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdf_tbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}

void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\\n");
}

#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex12(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}

#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.15 tut\_ex13\_1.c

Defining a table with a data structure for the table.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#else
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#endif
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdpf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (defined _WIN32 || defined __WIN32__)
        if (0==r)
            snprintf(buf, sizeof buf, "Header %02ix%02i", r, c);
        else
            snprintf(buf, sizeof buf, "Content %02ix%02i", r, c);
    #else
        if (0==r)
            snprintf(buf, sizeof buf, "Header %02zux%02zu", r, c);
        else
            snprintf(buf, sizeof buf, "Content %02zux%02zu", r, c);
    #endif
    return buf;
}
static char *
cb_label(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (defined _WIN32 || defined __WIN32__)
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %ix%i:", r, c);
        }
    #else
        if (0==r && 0==c) {
            snprintf(buf, sizeof buf, "Date:");
        } else {
            snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
        }
    #endif
    return buf;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
hpdpftbl_spec_t tbl_spec = {
    // Title and header flag
    .title=NULL, .use_header=TRUE,
    // Label and labelgrid flags
    .use_labels=FALSE, .use_labelgrid=FALSE,
    // Row and columns
    .rows=4, .cols=3,
    // xpos and ypos
    .xpos=hpdpftbl_cm2dpi(1), .ypos=hpdpftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
}
```

```

        .width=hpdtbtl_cm2dpi(15), .height=0,
        // Content and label callback
        .content_cb=cb_content, .label_cb=cb_label,
        // Style and table post creation callback
        .style_cb=NULL, .post_cb=NULL,
        // Pointer to optional cell specifications
        .cell_spec=NULL
    };
    void
    create_table_ex13_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
        hpdtbtl_stroke_from_data(pdf_doc, pdf_page, &tbt_spec, NULL);
    }
    // Setup a new PDF document with one page
    void
    setup_hpdtbtl(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
        // Setup the basic PDF document
        *pdf_doc = HPDF_New(error_handler, NULL);
        *pdf_page = HPDF_AddPage(*pdf_doc);
        HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
        HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
        if (addgrid) {
            hpdtbtl_stroke_grid(*pdf_doc, *pdf_page);
        }
    }
    void
    stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
        printf("Sending to file \"%s\" ...\\n", file);
        if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
            fprintf(stderr, "ERROR: Cannot save to file!");
        }
        HPDF_Free(pdf_doc);
        printf("Done.\\n");
    }
#ifdef _MSC_VER
    // Silent gcc about unused "arg" in the callback and error functions
    #pragma GCC diagnostic push
    #pragma GCC diagnostic ignored "-Wunused-parameter"
    #endif
    int
    main(int argc, char **argv) {
        HPDF_Doc pdf_doc;
        HPDF_Page pdf_page;
        if (setjmp(env)) {
            HPDF_Free(pdf_doc);
            return EXIT_FAILURE;
        }
        setup_hpdtbtl(&pdf_doc, &pdf_page, FALSE);
        create_table_ex13_1(pdf_doc, pdf_page);
        if ( 2==argc ) {
            struct stat sb;
            if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
                stroke_pdfdoc(pdf_doc, argv[1]);
                return EXIT_SUCCESS;
            }
        }
        char fname[255];
        snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
        stroke_pdfdoc(pdf_doc, fname);
        return EXIT_SUCCESS;
    }
#ifdef _MSC_VER
    #pragma GCC diagnostic pop
    #endif

```

## 16.16 tut\_ex13\_2.c

Defining a table with a data structure for table and cells.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdtbtl.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)

```

```

#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
//static char *
//cb_date(void *tag, size_t r, size_t c) {
//    static char buf[64];
//    time_t t = time(NULL);
//    ctime_r(&t, buf);
//    return buf;
//}
static char *
cb_content(void *tag, size_t r, size_t c) {
    static char *cell_content[] =
        {"Mark Ericson",
         "12 Sep 2021",
         "123 Downer Mews",
         "London",
         "NW2 HB3",
         "mark.p.ericson@myfinemail.com",
         "+44734 354 184 56",
         "+44771 938 137 11"};
    if (0==r && 0==c) return cell_content[0];
    else if (0==r && 3==c) return cell_content[1];
    else if (1==r && 0==c) return cell_content[2];
    else if (2==r && 0==c) return cell_content[3];
    else if (2==r && 3==c) return cell_content[4];
    else if (3==r && 0==c) return cell_content[5];
    else if (4==r && 0==c) return cell_content[6];
    else if (4==r && 2==c) return cell_content[7];
    else return NULL;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
hpdftbl_cell_spec_t cell_specs[] = {
    {.row=0, .col=0, .rowspan=1, .colspan=3,
     .label="Name:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=0, .col=3, .rowspan=1, .colspan=1,
     .label="Date:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=1, .col=0, .rowspan=1, .colspan=4,
     .label="Address:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=2, .col=0, .rowspan=1, .colspan=3,
     .label="City:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=2, .col=3, .rowspan=1, .colspan=1,
     .label="Zip:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=3, .col=0, .rowspan=1, .colspan=4,
     .label="E-mail:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=4, .col=0, .rowspan=1, .colspan=2,
     .label="Work-phone:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    {.row=4, .col=2, .rowspan=1, .colspan=2,
     .label="Mobile:",
     .content_cb=NULL, .label_cb=NULL, .style_cb=NULL, .canvas_cb=NULL},
    HPDF_TBL_END_CELLSPECS
};
hpdftbl_spec_t tbl_spec = {
    // Title and header flag
    .title=NULL, .use_header=FALSE,
    // Label and labelgrid flags
    .use_labels=TRUE, .use_labelgrid=TRUE,
    // Row and columns
    .rows=5, .cols=4,

```

```

        // xpos and ypos
        .xpos=hpdfdbl_cm2dpi(1), .ypos=hpdfdbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
        // width and height
        .width=hpdfdbl_cm2dpi(15), .height=0,
        // Content and label callback
        .content_cb=cb_content, .label_cb=0,
        // Style and table post creation callback
        .style_cb=NULL, .post_cb=NULL,
        // Pointer to optional cell specifications
        .cell_spec=cell_specs
    };
    void
    create_table_ex13_2(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
        hpdfdbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
    }
    // Setup a new PDF document with one page
    void
    setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
        // Setup the basic PDF document
        *pdf_doc = HPDF_New(error_handler, NULL);
        *pdf_page = HPDF_AddPage(*pdf_doc);
        HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
        HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
        if (addgrid) {
            hpdfdbl_stroke_grid(*pdf_doc, *pdf_page);
        }
    }
    void
    stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
        printf("Sending to file \"%s\" ...\\n", file);
        if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
            fprintf(stderr, "ERROR: Cannot save to file!");
        }
        HPDF_Free(pdf_doc);
        printf("Done.\\n");
    }
#ifdef _MSC_VER
    // Silent gcc about unused "arg" in the callback and error functions
    #pragma GCC diagnostic push
    #pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
    int
    main(int argc, char **argv) {
        HPDF_Doc pdf_doc;
        HPDF_Page pdf_page;
        if (setjmp(env)) {
            HPDF_Free(pdf_doc);
            return EXIT_FAILURE;
        }
        setup_hpdf(&pdf_doc, &pdf_page, FALSE);
        create_table_ex13_2(pdf_doc, pdf_page);
        if (2==argc) {
            struct stat sb;
            if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
                stroke_pdfdoc(pdf_doc, argv[1]);
                return EXIT_SUCCESS;
            }
        }
        char fname[255];
        snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
        stroke_pdfdoc(pdf_doc, fname);
        return EXIT_SUCCESS;
    }
#ifdef _MSC_VER
    #pragma GCC diagnostic pop
#endif

```

## 16.17 tut\_ex14.c

Defining a table with widgets.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined __WIN32__ || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>

```

```

#include <setjmp.h>
#include <time.h>
#if !(defined __WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}

static char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Device name:");
    } else if (0==r && 1==c) {
        snprintf(buf, sizeof buf, "Date:");
    } else if (1==r && 0==c) {
        snprintf(buf, sizeof buf, "Battery strength:");
    } else if (1==r && 1==c) {
        snprintf(buf, sizeof buf, "Signal:");
    } else {
        return NULL;
    }
    return buf;
}

static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! static_date ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}

static char *
cb_device_name(void *tag, size_t r, size_t c) {
    static char buf[32];
    snprintf(buf, sizeof buf, "IoT Device ABC123");
    return buf;
}

void
cb_draw_battery_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                       size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                       HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL segment_tot_width = width * 0.5;
    const HPDF_REAL segment_height = height / 3;
    const HPDF_REAL segment_xpos = xpos + 40;
    const HPDF_REAL segment_ypos = ypos + 4;
    const size_t num_segments = 10;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_GREEN;
    const double val_percent = 0.4;
    const _Bool val_text_hide = FALSE;
    hpdftbl_widget_segment_hbar(
        doc, page, segment_xpos, segment_ypos, segment_tot_width,
        segment_height, num_segments, on_color, val_percent, val_text_hide);
}

void
cb_draw_signal_widget(HPDF_Doc doc, HPDF_Page page, void *tag, size_t r,
                      size_t c, HPDF_REAL xpos, HPDF_REAL ypos,
                      HPDF_REAL width, HPDF_REAL height) {
    const HPDF_REAL wwidth = 35;
    const HPDF_REAL wheight = 20;
    const HPDF_REAL wxpos = xpos + 70;
    const HPDF_REAL wypos = ypos + 4;
    const size_t num_segments = 5;
    const HPDF_RGBColor on_color = HPDF_COLOR_DARK_RED;
    // This should be the real data retrieved from a DB (for example)

```



```

    const size_t num_on_segments = 3;
    hpdf_tbl_widget_strength_meter(doc, page, wxpos, wypos, wwidth, wheight,
                                   num_segments, on_color, num_on_segments);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex14(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex14: 2x2 table widget callbacks";
    hpdf_tbl_t tbl = hpdf_tbl_create_title(num_rows, num_cols, table_title);
    hpdf_tbl_use_labels(tbl, TRUE);
    hpdf_tbl_use_labelgrid(tbl, TRUE);
    // Use one label callback for the entire table
    hpdf_tbl_set_label_cb(tbl, cb_labels);
    // Name in top left corner
    hpdf_tbl_set_cell_content_cb(tbl, 0, 0, cb_device_name);
    // Date in top right corner
    hpdf_tbl_set_cell_content_cb(tbl, 0, 1, cb_date);
    // Draw battery strength
    hpdf_tbl_set_cell_canvas_cb(tbl, 1, 0, cb_draw_battery_widget);
    // Draw signal strength
    hpdf_tbl_set_cell_canvas_cb(tbl, 1, 1, cb_draw_signal_widget);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(12);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpfd(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdf_tbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    setup_hpfd(&pdf_doc, &pdf_page, FALSE);
    create_table_ex14(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.18 tut\_ex15.c

Defining a table with zebra lines.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                        void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    //hpdftbl_use_header(tbl, TRUE);
    hpdftbl_set_zebra(tbl, TRUE, 0);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(18);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
```

```

void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ..\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {

    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc ;
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex15(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.19 tut\_ex15\_1.c

Defining a table with zebra lines and different phase.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For the case when we use this example as a unit/integration test
_Bool static_date = FALSE;
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#pragma GCC diagnostic ignored "-Wformat-nonliteral"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {

```

```

    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
        hpdf_tbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
typedef char **content_t;
void setup_dummy_data(content_t *content, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
#endif _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex15(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 7;
    const size_t num_cols = 5;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_data(&content, num_rows, num_cols);
    hpdf_tbl_set_content(tbl, content);
    //hpdf_tbl_use_header(tbl, TRUE);
    hpdf_tbl_set_zebra(tbl, TRUE, 1);
    // Normal inner line (same color as default Zebra to make them "invisible"
    hpdf_tbl_set_inner_hgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY, LINE_SOLID);
    // Top inner line. Comment this line to get a visible top line
    hpdf_tbl_set_inner_tgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY, LINE_SOLID);
    HPDF_REAL xpos = hpdf_tbl_cm2dpi(1);
    HPDF_REAL ypos = hpdf_tbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdf_tbl_cm2dpi(18);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdf_tbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#endif _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    // For the case when we use this example as a unit/integration test we need to
    // look down a static date since we cannot compare otherwise since the date
    // strings will be different.
    static_date = 2==argc;
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex15(pdf_doc, pdf_page);
    if (2==argc) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
}

```

```

    }
}
char fname[255];
snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
stroke_pdfdoc(pdf_doc, fname);
return EXIT_SUCCESS;
}
#endifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

## 16.20 tut\_ex20.c

Defining a table and adjusting the gridlines.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#if !(defined _WIN32 || defined __WIN32__)
#include <sys/utsname.h>
#include <libgen.h>
#include <sys/stat.h>
#endif
// This include should always be used
#include "../src/hpdftbl.h"
// For simulated exception handling
jmp_buf env;
#ifdef _MSC_VER
// Silent gcc about unused "arg" in the callback and error functions
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
// A standard hpdf error handler which also translates the hpdf error code to a
// human readable string
static void error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
                          void *user_data) {
    fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
            hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
    longjmp(env, 1);
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
typedef char **content_t;
void setup_dummy_data(content_t *content, content_t *labels, size_t rows, size_t cols) {
    char buff[255];
    *content = calloc(rows*cols, sizeof(char*));
    *labels = calloc(rows*cols, sizeof(char*));
    size_t cnt = 0;
    for (size_t r = 0; r < rows; r++) {
        for (size_t c = 0; c < cols; c++) {
            snprintf(buff, sizeof(buff), "Content %zu", cnt);
            (*content)[cnt] = strdup(buff);
            snprintf(buff, sizeof(buff), "Label %zu", cnt);
            (*labels)[cnt] = strdup(buff);
            cnt++;
        }
    }
}
void
create_table_ex20(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 5;
    const size_t num_cols = 4;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_data(&content, &labels, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_labels(tbl, labels);

    hpdftbl_use_labels(tbl, FALSE);
    hpdftbl_use_labelgrid(tbl, TRUE);
    hpdftbl_use_header(tbl, FALSE);
    hpdftbl_set_inner_vgrid_style(tbl, 0.7, HPDF_COLOR_DARK_GRAY, LINE_SOLID);
}

```

```

    hpdftbl_set_inner_hgrid_style(tbl, 0.8, HPDF_COLOR_GRAY, LINE_DOT1);
    hpdftbl_set_inner_tgrid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
    hpdftbl_set_outer_grid_style(tbl, 1.5, HPDF_COLOR_BLACK, LINE_SOLID);
    HPDF_REAL xpos = hpdftbl_cm2dpi(1);
    HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
    HPDF_REAL width = hpdftbl_cm2dpi(10);
    HPDF_REAL height = 0; // Calculate height automatically
    // Stroke the table to the page
    hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
// Setup a new PDF document with one page
void
setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
    // Setup the basic PDF document
    *pdf_doc = HPDF_New(error_handler, NULL);
    *pdf_page = HPDF_AddPage(*pdf_doc);
    HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
    HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
    if (addgrid) {
        hpdftbl_stroke_grid(*pdf_doc, *pdf_page);
    }
}
void
stroke_pdfdoc(HPDF_Doc pdf_doc, char *file) {
    printf("Sending to file \"%s\" ...\n", file);
    if (HPDF_OK != HPDF_SaveToFile(pdf_doc, file)) {
        fprintf(stderr, "ERROR: Cannot save to file!");
    }
    HPDF_Free(pdf_doc);
    printf("Done.\n");
}
#ifdef _MSC_VER
// Silent gcc about unused "arg"
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wunused-parameter"
#endif
int
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    if (setjmp(env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_ex20(pdf_doc, pdf_page);
    if ( 2==argc ) {
        struct stat sb;
        if (stat(dirname(argv[1]), &sb) == 0 && S_ISDIR(sb.st_mode)) {
            stroke_pdfdoc(pdf_doc, argv[1]);
            return EXIT_SUCCESS;
        }
    }
    char fname[255];
    snprintf(fname, sizeof fname, "out/%s.pdf", basename(argv[0]));
    stroke_pdfdoc(pdf_doc, fname);
    return EXIT_SUCCESS;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif

```

# Index

- `_HPDFTBL_SET_ERR`
    - `hpdtbl.h`, [126](#)
- `background`
  - `text_style`, [76](#)
- `bootstrap.sh`, [79](#)
- `bottom_vmargin_factor`
  - `hpdtbl`, [53](#)
  - `hpdtbl_theme`, [71](#)
- `canvas_cb`
  - `hpdtbl`, [53](#)
  - `hpdtbl_cell`, [61](#)
  - `hpdtbl_cell_spec`, [64](#)
- `cell_spec`
  - `hpdtbl_spec`, [67](#)
- `cells`
  - `hpdtbl`, [54](#)
- `CENTER`
  - `hpdtbl.h`, [132](#)
- `col`
  - `hpdtbl_cell_spec`, [64](#)
- `col_width_percent`
  - `hpdtbl`, [54](#)
- `color`
  - `grid_style`, [51](#)
  - `text_style`, [76](#)
- `cols`
  - `hpdtbl`, [54](#)
  - `hpdtbl_spec`, [67](#)
- `colspan`
  - `hpdtbl_cell`, [61](#)
  - `hpdtbl_cell_spec`, [64](#)
- `config.h`, [82](#)
- `content`
  - `hpdtbl_cell`, [61](#)
- `content_cb`
  - `hpdtbl`, [54](#)
  - `hpdtbl_cell`, [61](#)
  - `hpdtbl_cell_spec`, [64](#)
  - `hpdtbl_spec`, [68](#)
- `content_style`
  - `hpdtbl`, [54](#)
  - `hpdtbl_cell`, [61](#)
  - `hpdtbl_theme`, [71](#)
- `content_style_cb`
  - `hpdtbl`, [55](#)
- `dash_ptn`
  - `line_dash_style`, [75](#)
- `dbgbl.sh`, [79](#)
- `DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR`
  - `hpdtbl.h`, [127](#)
- `delta_x`
  - `hpdtbl_cell`, [61](#)
- `delta_y`
  - `hpdtbl_cell`, [62](#)
- `docupload.sh.in`, [80](#)
  - `GITHUB_USER`, [81](#)
  - `PDFFILE_COPY`, [81](#)
- `errcode`
  - `hpdtbl_errcode_entry`, [66](#)
- `errstr`
  - `hpdtbl_errcode_entry`, [66](#)
- `FALSE`
  - `hpdtbl_widget.c`, [183](#)
- `font`
  - `text_style`, [76](#)
- `fsize`
  - `text_style`, [77](#)
- `GITHUB_USER`
  - `docupload.sh.in`, [81](#)
- `grid_style`, [51](#)
  - `color`, [51](#)
  - `line_dashstyle`, [51](#)
  - `width`, [52](#)
- `halign`
  - `text_style`, [77](#)
- `header_style`
  - `hpdtbl`, [55](#)
  - `hpdtbl_theme`, [71](#)
- `height`
  - `hpdtbl`, [55](#)
  - `hpdtbl_cell`, [62](#)
  - `hpdtbl_spec`, [68](#)
- `HPDF_RoundedCornerRectangle`
  - `hpdtbl.c`, [87](#)
  - `hpdtbl.h`, [132](#)
- `hpdf_text_style_t`
  - `hpdtbl.h`, [127](#)
- `hpdtbl`, [52](#)
  - `bottom_vmargin_factor`, [53](#)
  - `canvas_cb`, [53](#)
  - `cells`, [54](#)
  - `col_width_percent`, [54](#)
  - `cols`, [54](#)

- content\_cb, 54
- content\_style, 54
- content\_style\_cb, 55
- header\_style, 55
- height, 55
- inner\_hgrid, 55
- inner\_tgrid, 55
- inner\_vgrid, 56
- label\_cb, 56
- label\_style, 56
- minheight, 56
- outer\_grid, 56
- pdf\_doc, 57
- pdf\_page, 57
- posx, 57
- posy, 57
- rows, 57
- tag, 57
- title\_style, 58
- title\_txt, 58
- use\_cell\_labels, 58
- use\_header\_row, 58
- use\_label\_grid\_style, 58
- use\_zebra, 59
- width, 59
- zebra\_color1, 59
- zebra\_color2, 59
- zebra\_phase, 60
- hpdfbtbl.c, 83
  - HPDF\_RoundedCornerRectangle, 87
  - hpdfbtbl\_clear\_spanning, 87
  - hpdfbtbl\_create, 88
  - hpdfbtbl\_create\_title, 88
  - hpdfbtbl\_default\_table\_error\_handler, 89
  - hpdfbtbl\_destroy, 90
  - hpdfbtbl\_encoding\_text\_out, 90
  - hpdfbtbl\_get\_anchor\_top\_left, 91
  - hpdfbtbl\_get\_errstr, 91
  - hpdfbtbl\_get\_last\_auto\_height, 91
  - hpdfbtbl\_get\_last\_errcode, 92
  - hpdfbtbl\_set\_anchor\_top\_left, 92
  - hpdfbtbl\_set\_background, 93
  - hpdfbtbl\_set\_bottom\_vmargin\_factor, 93
  - hpdfbtbl\_set\_canvas\_cb, 94
  - hpdfbtbl\_set\_cell, 94
  - hpdfbtbl\_set\_cell\_canvas\_cb, 95
  - hpdfbtbl\_set\_cell\_content\_cb, 95
  - hpdfbtbl\_set\_cell\_content\_style, 96
  - hpdfbtbl\_set\_cell\_content\_style\_cb, 97
  - hpdfbtbl\_set\_cell\_label\_cb, 98
  - hpdfbtbl\_set\_cellspan, 98
  - hpdfbtbl\_set\_col\_content\_style, 99
  - hpdfbtbl\_set\_colwidth\_percent, 100
  - hpdfbtbl\_set\_content, 100
  - hpdfbtbl\_set\_content\_cb, 101
  - hpdfbtbl\_set\_content\_style, 101
  - hpdfbtbl\_set\_content\_style\_cb, 102
  - hpdfbtbl\_set\_errhandler, 103
  - hpdfbtbl\_set\_header\_halign, 103
  - hpdfbtbl\_set\_header\_style, 104
  - hpdfbtbl\_set\_inner\_grid\_style, 105
  - hpdfbtbl\_set\_inner\_hgrid\_style, 105
  - hpdfbtbl\_set\_inner\_tgrid\_style, 106
  - hpdfbtbl\_set\_inner\_vgrid\_style, 106
  - hpdfbtbl\_set\_label\_cb, 107
  - hpdfbtbl\_set\_label\_style, 108
  - hpdfbtbl\_set\_labels, 108
  - hpdfbtbl\_set\_line\_dash, 109
  - hpdfbtbl\_set\_min\_rowheight, 110
  - hpdfbtbl\_set\_outer\_grid\_style, 110
  - hpdfbtbl\_set\_row\_content\_style, 111
  - hpdfbtbl\_set\_tag, 111
  - hpdfbtbl\_set\_text\_encoding, 112
  - hpdfbtbl\_set\_title, 112
  - hpdfbtbl\_set\_title\_halign, 114
  - hpdfbtbl\_set\_title\_style, 114
  - hpdfbtbl\_set\_zebra, 115
  - hpdfbtbl\_set\_zebra\_color, 116
  - hpdfbtbl\_stroke, 116
  - hpdfbtbl\_stroke\_from\_data, 117
  - hpdfbtbl\_use\_header, 118
  - hpdfbtbl\_use\_labelgrid, 118
  - hpdfbtbl\_use\_labels, 119
- hpdfbtbl.h, 120, 169
  - \_HPDFTBL\_SET\_ERR, 126
  - CENTER, 132
  - DEFAULT\_AUTO\_VBOTTOM\_MARGIN\_FACTOR, 127
  - HPDF\_RoundedCornerRectangle, 132
  - hpdf\_text\_style\_t, 127
  - hpdfbtbl\_apply\_theme, 132
  - hpdfbtbl\_callback\_t, 127
  - hpdfbtbl\_canvas\_callback\_t, 128
  - hpdfbtbl\_cell\_spec\_t, 128
  - hpdfbtbl\_cell\_t, 128
  - hpdfbtbl\_clear\_spanning, 133
  - hpdfbtbl\_cm2dpi, 127
  - hpdfbtbl\_content\_callback\_t, 128
  - hpdfbtbl\_content\_style\_callback\_t, 129
  - hpdfbtbl\_create, 133
  - hpdfbtbl\_create\_title, 134
  - hpdfbtbl\_dashstyle, 131
  - hpdfbtbl\_default\_table\_error\_handler, 134
  - hpdfbtbl\_destroy, 135
  - hpdfbtbl\_destroy\_theme, 135
  - hpdfbtbl\_encoding\_text\_out, 136
  - hpdfbtbl\_error\_handler\_t, 129
  - hpdfbtbl\_get\_anchor\_top\_left, 136
  - hpdfbtbl\_get\_default\_theme, 136
  - hpdfbtbl\_get\_errstr, 137
  - hpdfbtbl\_get\_last\_auto\_height, 137
  - hpdfbtbl\_get\_last\_errcode, 138
  - hpdfbtbl\_grid\_style\_t, 129
  - hpdfbtbl\_hpdl\_get\_errstr, 138
  - hpdfbtbl\_line\_dashstyle\_t, 129
  - hpdfbtbl\_set\_anchor\_top\_left, 139



[hpdfctl\\_set\\_background](#), 139  
[hpdfctl\\_set\\_bottom\\_vmargin\\_factor](#), 140  
[hpdfctl\\_set\\_canvas\\_cb](#), 140  
[hpdfctl\\_set\\_cell](#), 141  
[hpdfctl\\_set\\_cell\\_canvas\\_cb](#), 141  
[hpdfctl\\_set\\_cell\\_content\\_cb](#), 142  
[hpdfctl\\_set\\_cell\\_content\\_style](#), 142  
[hpdfctl\\_set\\_cell\\_content\\_style\\_cb](#), 143  
[hpdfctl\\_set\\_cell\\_label\\_cb](#), 144  
[hpdfctl\\_set\\_cellspan](#), 144  
[hpdfctl\\_set\\_col\\_content\\_style](#), 145  
[hpdfctl\\_set\\_colwidth\\_percent](#), 146  
[hpdfctl\\_set\\_content](#), 146  
[hpdfctl\\_set\\_content\\_cb](#), 147  
[hpdfctl\\_set\\_content\\_style](#), 147  
[hpdfctl\\_set\\_content\\_style\\_cb](#), 148  
[hpdfctl\\_set\\_errhandler](#), 149  
[hpdfctl\\_set\\_header\\_halign](#), 149  
[hpdfctl\\_set\\_header\\_style](#), 150  
[hpdfctl\\_set\\_inner\\_grid\\_style](#), 150  
[hpdfctl\\_set\\_inner\\_hgrid\\_style](#), 151  
[hpdfctl\\_set\\_inner\\_tgrid\\_style](#), 151  
[hpdfctl\\_set\\_inner\\_vgrid\\_style](#), 152  
[hpdfctl\\_set\\_label\\_cb](#), 153  
[hpdfctl\\_set\\_label\\_style](#), 153  
[hpdfctl\\_set\\_labels](#), 154  
[hpdfctl\\_set\\_min\\_rowheight](#), 154  
[hpdfctl\\_set\\_outer\\_grid\\_style](#), 155  
[hpdfctl\\_set\\_row\\_content\\_style](#), 155  
[hpdfctl\\_set\\_tag](#), 156  
[hpdfctl\\_set\\_text\\_encoding](#), 157  
[hpdfctl\\_set\\_title](#), 157  
[hpdfctl\\_set\\_title\\_halign](#), 157  
[hpdfctl\\_set\\_title\\_style](#), 158  
[hpdfctl\\_set\\_zebra](#), 159  
[hpdfctl\\_set\\_zebra\\_color](#), 159  
[hpdfctl\\_spec\\_t](#), 130  
[hpdfctl\\_stroke](#), 160  
[hpdfctl\\_stroke\\_from\\_data](#), 160  
[hpdfctl\\_stroke\\_grid](#), 161  
[hpdfctl\\_t](#), 130  
[hpdfctl\\_table\\_widget\\_letter\\_buttons](#), 162  
[hpdfctl\\_text\\_align](#), 131  
[hpdfctl\\_text\\_align\\_t](#), 130  
[hpdfctl\\_theme\\_t](#), 130  
[hpdfctl\\_use\\_header](#), 162  
[hpdfctl\\_use\\_labelgrid](#), 164  
[hpdfctl\\_use\\_labels](#), 164  
[hpdfctl\\_widget\\_hbar](#), 166  
[hpdfctl\\_widget\\_segment\\_hbar](#), 167  
[hpdfctl\\_widget\\_slide\\_button](#), 167  
[hpdfctl\\_widget\\_strength\\_meter](#), 168  
[LEFT](#), 132  
[LINE\\_DASH1](#), 131  
[LINE\\_DASH2](#), 131  
[LINE\\_DASH3](#), 131  
[LINE\\_DASH4](#), 131  
[LINE\\_DASHDOT1](#), 131  
[LINE\\_DASHDOT2](#), 131  
[LINE\\_DOT1](#), 131  
[LINE\\_DOT2](#), 131  
[LINE\\_DOT3](#), 131  
[LINE\\_SOLID](#), 131  
[RIGHT](#), 132  
[hpdfctl\\_apply\\_theme](#)  
[hpdfctl.h](#), 132  
[hpdfctl\\_theme.c](#), 180  
[hpdfctl\\_callback\\_t](#)  
[hpdfctl.h](#), 127  
[hpdfctl\\_canvas\\_callback\\_t](#)  
[hpdfctl.h](#), 128  
[hpdfctl\\_cell](#), 60  
[canvas\\_cb](#), 61  
[colspan](#), 61  
[content](#), 61  
[content\\_cb](#), 61  
[content\\_style](#), 61  
[delta\\_x](#), 61  
[delta\\_y](#), 62  
[height](#), 62  
[label](#), 62  
[label\\_cb](#), 62  
[parent\\_cell](#), 62  
[rowspan](#), 62  
[style\\_cb](#), 63  
[textwidth](#), 63  
[width](#), 63  
[hpdfctl\\_cell\\_spec](#), 63  
[canvas\\_cb](#), 64  
[col](#), 64  
[colspan](#), 64  
[content\\_cb](#), 64  
[label](#), 65  
[label\\_cb](#), 65  
[row](#), 65  
[rowspan](#), 65  
[style\\_cb](#), 65  
[hpdfctl\\_cell\\_spec\\_t](#)  
[hpdfctl.h](#), 128  
[hpdfctl\\_cell\\_t](#)  
[hpdfctl.h](#), 128  
[hpdfctl\\_clear\\_spanning](#)  
[hpdfctl.c](#), 87  
[hpdfctl.h](#), 133  
[hpdfctl\\_cm2dpi](#)  
[hpdfctl.h](#), 127  
[hpdfctl\\_content\\_callback\\_t](#)  
[hpdfctl.h](#), 128  
[hpdfctl\\_content\\_style\\_callback\\_t](#)  
[hpdfctl.h](#), 129  
[hpdfctl\\_create](#)  
[hpdfctl.c](#), 88  
[hpdfctl.h](#), 133  
[hpdfctl\\_create\\_title](#)  
[hpdfctl.c](#), 88  
[hpdfctl.h](#), 134

hpdfctl\_dashstyle  
     hpdfctl.h, 131  
 HPDFTBL\_DEFAULT\_CONTENT\_STYLE  
     hpdfctl\_theme.c, 178  
 HPDFTBL\_DEFAULT\_HEADER\_STYLE  
     hpdfctl\_theme.c, 178  
 HPDFTBL\_DEFAULT\_INNER\_HGRID\_STYLE  
     hpdfctl\_theme.c, 179  
 HPDFTBL\_DEFAULT\_INNER\_VGRID\_STYLE  
     hpdfctl\_theme.c, 179  
 HPDFTBL\_DEFAULT\_LABEL\_STYLE  
     hpdfctl\_theme.c, 179  
 HPDFTBL\_DEFAULT\_OUTER\_GRID\_STYLE  
     hpdfctl\_theme.c, 179  
 hpdfctl\_default\_table\_error\_handler  
     hpdfctl.c, 89  
     hpdfctl.h, 134  
 HPDFTBL\_DEFAULT\_ZEBRA\_COLOR1  
     hpdfctl\_theme.c, 180  
 HPDFTBL\_DEFAULT\_ZEBRA\_COLOR2  
     hpdfctl\_theme.c, 180  
 hpdfctl\_destroy  
     hpdfctl.c, 90  
     hpdfctl.h, 135  
 hpdfctl\_destroy\_theme  
     hpdfctl.h, 135  
     hpdfctl\_theme.c, 181  
 hpdfctl\_encoding\_text\_out  
     hpdfctl.c, 90  
     hpdfctl.h, 136  
 hpdfctl\_errcode\_entry, 66  
     errcode, 66  
     errstr, 66  
 hpdfctl\_error\_handler\_t  
     hpdfctl.h, 129  
 hpdfctl\_errstr.c, 175  
     hpdfctl\_hpdc\_get\_errstr, 176  
 hpdfctl\_get\_anchor\_top\_left  
     hpdfctl.c, 91  
     hpdfctl.h, 136  
 hpdfctl\_get\_default\_theme  
     hpdfctl.h, 136  
     hpdfctl\_theme.c, 181  
 hpdfctl\_get\_errstr  
     hpdfctl.c, 91  
     hpdfctl.h, 137  
 hpdfctl\_get\_last\_auto\_height  
     hpdfctl.c, 91  
     hpdfctl.h, 137  
 hpdfctl\_get\_last\_errcode  
     hpdfctl.c, 92  
     hpdfctl.h, 138  
 hpdfctl\_grid.c, 176  
     hpdfctl\_stroke\_grid, 176  
 hpdfctl\_grid\_style\_t  
     hpdfctl.h, 129  
 hpdfctl\_hpdc\_get\_errstr  
     hpdfctl.h, 138  
     hpdfctl\_errstr.c, 176  
 hpdfctl\_line\_dashstyle\_t  
     hpdfctl.h, 129  
 hpdfctl\_set\_anchor\_top\_left  
     hpdfctl.c, 92  
     hpdfctl.h, 139  
 hpdfctl\_set\_background  
     hpdfctl.c, 93  
     hpdfctl.h, 139  
 hpdfctl\_set\_bottom\_vmargin\_factor  
     hpdfctl.c, 93  
     hpdfctl.h, 140  
 hpdfctl\_set\_canvas\_cb  
     hpdfctl.c, 94  
     hpdfctl.h, 140  
 hpdfctl\_set\_cell  
     hpdfctl.c, 94  
     hpdfctl.h, 141  
 hpdfctl\_set\_cell\_canvas\_cb  
     hpdfctl.c, 95  
     hpdfctl.h, 141  
 hpdfctl\_set\_cell\_content\_cb  
     hpdfctl.c, 95  
     hpdfctl.h, 142  
 hpdfctl\_set\_cell\_content\_style  
     hpdfctl.c, 96  
     hpdfctl.h, 142  
 hpdfctl\_set\_cell\_content\_style\_cb  
     hpdfctl.c, 97  
     hpdfctl.h, 143  
 hpdfctl\_set\_cell\_label\_cb  
     hpdfctl.c, 98  
     hpdfctl.h, 144  
 hpdfctl\_set\_cellspan  
     hpdfctl.c, 98  
     hpdfctl.h, 144  
 hpdfctl\_set\_col\_content\_style  
     hpdfctl.c, 99  
     hpdfctl.h, 145  
 hpdfctl\_set\_colwidth\_percent  
     hpdfctl.c, 100  
     hpdfctl.h, 146  
 hpdfctl\_set\_content  
     hpdfctl.c, 100  
     hpdfctl.h, 146  
 hpdfctl\_set\_content\_cb  
     hpdfctl.c, 101  
     hpdfctl.h, 147  
 hpdfctl\_set\_content\_style  
     hpdfctl.c, 101  
     hpdfctl.h, 147  
 hpdfctl\_set\_content\_style\_cb  
     hpdfctl.c, 102  
     hpdfctl.h, 148  
 hpdfctl\_set\_errhandler  
     hpdfctl.c, 103  
     hpdfctl.h, 149  
 hpdfctl\_set\_header\_halign

- hpdfctl.c, [103](#)
- hpdfctl.h, [149](#)
- hpdfctl\_set\_header\_style
  - hpdfctl.c, [104](#)
  - hpdfctl.h, [150](#)
- hpdfctl\_set\_inner\_grid\_style
  - hpdfctl.c, [105](#)
  - hpdfctl.h, [150](#)
- hpdfctl\_set\_inner\_hgrid\_style
  - hpdfctl.c, [105](#)
  - hpdfctl.h, [151](#)
- hpdfctl\_set\_inner\_tgrid\_style
  - hpdfctl.c, [106](#)
  - hpdfctl.h, [151](#)
- hpdfctl\_set\_inner\_vgrid\_style
  - hpdfctl.c, [106](#)
  - hpdfctl.h, [152](#)
- hpdfctl\_set\_label\_cb
  - hpdfctl.c, [107](#)
  - hpdfctl.h, [153](#)
- hpdfctl\_set\_label\_style
  - hpdfctl.c, [108](#)
  - hpdfctl.h, [153](#)
- hpdfctl\_set\_labels
  - hpdfctl.c, [108](#)
  - hpdfctl.h, [154](#)
- hpdfctl\_set\_line\_dash
  - hpdfctl.c, [109](#)
- hpdfctl\_set\_min\_rowheight
  - hpdfctl.c, [110](#)
  - hpdfctl.h, [154](#)
- hpdfctl\_set\_outer\_grid\_style
  - hpdfctl.c, [110](#)
  - hpdfctl.h, [155](#)
- hpdfctl\_set\_row\_content\_style
  - hpdfctl.c, [111](#)
  - hpdfctl.h, [155](#)
- hpdfctl\_set\_tag
  - hpdfctl.c, [111](#)
  - hpdfctl.h, [156](#)
- hpdfctl\_set\_text\_encoding
  - hpdfctl.c, [112](#)
  - hpdfctl.h, [157](#)
- hpdfctl\_set\_title
  - hpdfctl.c, [112](#)
  - hpdfctl.h, [157](#)
- hpdfctl\_set\_title\_halign
  - hpdfctl.c, [114](#)
  - hpdfctl.h, [157](#)
- hpdfctl\_set\_title\_style
  - hpdfctl.c, [114](#)
  - hpdfctl.h, [158](#)
- hpdfctl\_set\_zebra
  - hpdfctl.c, [115](#)
  - hpdfctl.h, [159](#)
- hpdfctl\_set\_zebra\_color
  - hpdfctl.c, [116](#)
  - hpdfctl.h, [159](#)
- hpdfctl\_spec, [67](#)
  - cell\_spec, [67](#)
  - cols, [67](#)
  - content\_cb, [68](#)
  - height, [68](#)
  - label\_cb, [68](#)
  - post\_cb, [68](#)
  - rows, [68](#)
  - style\_cb, [69](#)
  - title, [69](#)
  - use\_header, [69](#)
  - use\_labelgrid, [69](#)
  - use\_labels, [69](#)
  - width, [70](#)
  - xpos, [70](#)
  - ypos, [70](#)
- hpdfctl\_spec\_t
  - hpdfctl.h, [130](#)
- hpdfctl\_stroke
  - hpdfctl.c, [116](#)
  - hpdfctl.h, [160](#)
- hpdfctl\_stroke\_from\_data
  - hpdfctl.c, [117](#)
  - hpdfctl.h, [160](#)
- hpdfctl\_stroke\_grid
  - hpdfctl.h, [161](#)
  - hpdfctl\_grid.c, [176](#)
- hpdfctl\_t
  - hpdfctl.h, [130](#)
- hpdfctl\_table\_widget\_letter\_buttons
  - hpdfctl.h, [162](#)
  - hpdfctl\_widget.c, [183](#)
- hpdfctl\_text\_align
  - hpdfctl.h, [131](#)
- hpdfctl\_text\_align\_t
  - hpdfctl.h, [130](#)
- hpdfctl\_theme, [70](#)
  - bottom\_vmargin\_factor, [71](#)
  - content\_style, [71](#)
  - header\_style, [71](#)
  - inner\_hborder, [72](#)
  - inner\_tborder, [72](#)
  - inner\_vborder, [72](#)
  - label\_style, [72](#)
  - outer\_border, [72](#)
  - title\_style, [73](#)
  - use\_header\_row, [73](#)
  - use\_label\_grid\_style, [73](#)
  - use\_labels, [73](#)
  - use\_zebra, [73](#)
  - zebra\_color1, [74](#)
  - zebra\_color2, [74](#)
  - zebra\_phase, [74](#)
- hpdfctl\_theme.c, [177](#)
  - hpdfctl\_apply\_theme, [180](#)
  - HPDFTBL\_DEFAULT\_CONTENT\_STYLE, [178](#)
  - HPDFTBL\_DEFAULT\_HEADER\_STYLE, [178](#)

- HPDFTBL\_DEFAULT\_INNER\_HGRID\_STYLE, 179
- HPDFTBL\_DEFAULT\_INNER\_VGRID\_STYLE, 179
- HPDFTBL\_DEFAULT\_LABEL\_STYLE, 179
- HPDFTBL\_DEFAULT\_OUTER\_GRID\_STYLE, 179
- HPDFTBL\_DEFAULT\_ZEBRA\_COLOR1, 180
- HPDFTBL\_DEFAULT\_ZEBRA\_COLOR2, 180
- hpdfctl\_destroy\_theme, 181
- hpdfctl\_get\_default\_theme, 181
- hpdfctl\_theme\_t
  - hpdfctl.h, 130
- hpdfctl\_use\_header
  - hpdfctl.c, 118
  - hpdfctl.h, 162
- hpdfctl\_use\_labelgrid
  - hpdfctl.c, 118
  - hpdfctl.h, 164
- hpdfctl\_use\_labels
  - hpdfctl.c, 119
  - hpdfctl.h, 164
- hpdfctl\_widget.c, 182
  - FALSE, 183
  - hpdfctl\_table\_widget\_letter\_buttons, 183
  - hpdfctl\_widget\_hbar, 184
  - hpdfctl\_widget\_segment\_hbar, 184
  - hpdfctl\_widget\_slide\_button, 185
  - hpdfctl\_widget\_strength\_meter, 186
  - TRUE, 183
- hpdfctl\_widget\_hbar
  - hpdfctl.h, 166
  - hpdfctl\_widget.c, 184
- hpdfctl\_widget\_segment\_hbar
  - hpdfctl.h, 167
  - hpdfctl\_widget.c, 184
- hpdfctl\_widget\_slide\_button
  - hpdfctl.h, 167
  - hpdfctl\_widget.c, 185
- hpdfctl\_widget\_strength\_meter
  - hpdfctl.h, 168
  - hpdfctl\_widget.c, 186
- inner\_hborder
  - hpdfctl\_theme, 72
- inner\_hgrid
  - hpdfctl, 55
- inner\_tborder
  - hpdfctl\_theme, 72
- inner\_tgrid
  - hpdfctl, 55
- inner\_vborder
  - hpdfctl\_theme, 72
- inner\_vgrid
  - hpdfctl, 56
- label
  - hpdfctl\_cell, 62
  - hpdfctl\_cell\_spec, 65
- label\_cb
  - hpdfctl, 56
  - hpdfctl\_cell, 62
  - hpdfctl\_cell\_spec, 65
  - hpdfctl\_spec, 68
- label\_style
  - hpdfctl, 56
  - hpdfctl\_theme, 72
- LEFT
  - hpdfctl.h, 132
- LINE\_DASH1
  - hpdfctl.h, 131
- LINE\_DASH2
  - hpdfctl.h, 131
- LINE\_DASH3
  - hpdfctl.h, 131
- LINE\_DASH4
  - hpdfctl.h, 131
- line\_dash\_style, 74
  - dash\_ptn, 75
  - num, 75
- LINE\_DASHDOT1
  - hpdfctl.h, 131
- LINE\_DASHDOT2
  - hpdfctl.h, 131
- line\_dashstyle
  - grid\_style, 51
- LINE\_DOT1
  - hpdfctl.h, 131
- LINE\_DOT2
  - hpdfctl.h, 131
- LINE\_DOT3
  - hpdfctl.h, 131
- LINE\_SOLID
  - hpdfctl.h, 131
- minheight
  - hpdfctl, 56
- num
  - line\_dash\_style, 75
- outer\_border
  - hpdfctl\_theme, 72
- outer\_grid
  - hpdfctl, 56
- parent\_cell
  - hpdfctl\_cell, 62
- pdf\_doc
  - hpdfctl, 57
- pdf\_page
  - hpdfctl, 57
- PDFFILE\_COPY
  - docupload.sh.in, 81
- post\_cb
  - hpdfctl\_spec, 68
- posx
  - hpdfctl, 57
- posy

- hpdftbl, [57](#)
- RIGHT
  - hpdftbl.h, [132](#)
- row
  - hpdftbl\_cell\_spec, [65](#)
- rows
  - hpdftbl, [57](#)
  - hpdftbl\_spec, [68](#)
- rowspan
  - hpdftbl\_cell, [62](#)
  - hpdftbl\_cell\_spec, [65](#)
- stdbld.sh, [81](#)
- style\_cb
  - hpdftbl\_cell, [63](#)
  - hpdftbl\_cell\_spec, [65](#)
  - hpdftbl\_spec, [69](#)
- tag
  - hpdftbl, [57](#)
- text\_style, [75](#)
  - background, [76](#)
  - color, [76](#)
  - font, [76](#)
  - fsize, [77](#)
  - halign, [77](#)
- textwidth
  - hpdftbl\_cell, [63](#)
- title
  - hpdftbl\_spec, [69](#)
- title\_style
  - hpdftbl, [58](#)
  - hpdftbl\_theme, [73](#)
- title\_txt
  - hpdftbl, [58](#)
- TRUE
  - hpdftbl\_widget.c, [183](#)
- use\_cell\_labels
  - hpdftbl, [58](#)
- use\_header
  - hpdftbl\_spec, [69](#)
- use\_header\_row
  - hpdftbl, [58](#)
  - hpdftbl\_theme, [73](#)
- use\_label\_grid\_style
  - hpdftbl, [58](#)
  - hpdftbl\_theme, [73](#)
- use\_labelgrid
  - hpdftbl\_spec, [69](#)
- use\_labels
  - hpdftbl\_spec, [69](#)
  - hpdftbl\_theme, [73](#)
- use\_zebra
  - hpdftbl, [59](#)
  - hpdftbl\_theme, [73](#)
- width
  - grid\_style, [52](#)
  - hpdftbl, [59](#)
  - hpdftbl\_cell, [63](#)
  - hpdftbl\_spec, [70](#)
- xpos
  - hpdftbl\_spec, [70](#)
- ypos
  - hpdftbl\_spec, [70](#)
- zebra\_color1
  - hpdftbl, [59](#)
  - hpdftbl\_theme, [74](#)
- zebra\_color2
  - hpdftbl, [59](#)
  - hpdftbl\_theme, [74](#)
- zebra\_phase
  - hpdftbl, [60](#)
  - hpdftbl\_theme, [74](#)