

libhpdtbl

Generated on Sun May 15 2022 21:42:02 for libhpdtbl by Doxygen 1.9.3

Sun May 15 2022 21:42:02

1 Overview	1
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
2 Building the library	5
2.1 The short version; TL; DR	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><tt>libiconv</tt></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
3 Getting started	11
3.1 Creating an infrastructure for the examples	11
3.2 Your first table	12
3.3 Your second table - disconnecting program structure from data	14
3.4 Adding a header row	15
3.5 Using labels in the table cells	15
3.6 Adding a table title	16
3.7 Adjusting fonts and colors	17
4 Cell and row spanning	19
4.1 Cell and row spanning	19
4.2 Adjusting column width	19
5 Using callbacks	21
5.1 Introducing content callback functions	21
5.2 A content callback example	22
5.3 Dynamic (late binding) callbacks	23
5.3.1 Using late binding	24

6 Error handling	27
6.1 Using emulated exception handling	28
6.2 Additional information	29
6.3 Translating HPDF error codes	29
6.4 Example of setting up error handler	30
7 Font and style setting	31
7.1 Adjusting fonts and colors	31
7.2 Using style callbacks	32
7.2.1 Style callback example	33
7.3 Using style themes	34
7.4 Adjusting grid line styles	35
7.5 Adding zebra lines in a table	36
8 Tables layout from data	39
8.1 Defining a table in data	39
8.2 A first example of defining table as data	40
8.3 A second example of defining a table as data	40
9 Widgets	43
9.1 Overview	43
9.1.1 1. Segmented horizontal bar example	43
9.1.2 2. Horizontal bar example	43
9.1.3 3. Signal strength meter example	43
9.1.4 4. Radio sliding button example	44
9.1.5 5. Boxed letters example	44
9.2 Widget functions	44
9.2.1 Segmented horizontal bar defining function	44
9.2.2 Horizontal bar defining function	44
9.2.3 Signal strength defining function	45
9.2.4 Radio sliding button defining function	45
9.2.5 Boxed letters defining function	45
9.3 Usage	45
10 API Overview	47
10.1 Table creation related functions	47
10.2 Table error handling	47
10.3 Theme handling methods	47
10.4 Table layout adjusting functions	48
10.5 Table style modifying functions	48
10.6 Content handling	49
10.7 Callback handling	49
10.8 Dynamic (late binding) callback handling	49
10.9 Text encoding	50

10.10 Misc utility function	50
11 Todo List	51
12 Data Structure Index	53
12.1 Data Structures	53
13 File Index	55
13.1 File List	55
14 Data Structure Documentation	57
14.1 grid_style Struct Reference	57
14.1.1 Detailed Description	57
14.1.2 Field Documentation	57
14.1.2.1 color	57
14.1.2.2 line_dashstyle	58
14.1.2.3 width	58
14.2 hpdftbl Struct Reference	58
14.2.1 Detailed Description	59
14.2.2 Field Documentation	59
14.2.2.1 bottom_vmargin_factor	59
14.2.2.2 canvas_cb	60
14.2.2.3 cells	60
14.2.2.4 col_width_percent	60
14.2.2.5 cols	60
14.2.2.6 content_cb	60
14.2.2.7 content_style	61
14.2.2.8 content_style_cb	61
14.2.2.9 header_style	61
14.2.2.10 height	61
14.2.2.11 inner_hgrid	61
14.2.2.12 inner_tgrid	62
14.2.2.13 inner_vgrid	62
14.2.2.14 label_cb	62
14.2.2.15 label_style	62
14.2.2.16 minheight	62
14.2.2.17 outer_grid	63
14.2.2.18 pdf_doc	63
14.2.2.19 pdf_page	63
14.2.2.20 posx	63
14.2.2.21 posy	63
14.2.2.22 rows	63
14.2.2.23 tag	64
14.2.2.24 title_style	64

14.2.2.25 title_txt	64
14.2.2.26 use_cell_labels	64
14.2.2.27 use_header_row	64
14.2.2.28 use_label_grid_style	65
14.2.2.29 use_zebra	65
14.2.2.30 width	65
14.2.2.31 zebra_color1	65
14.2.2.32 zebra_color2	66
14.2.2.33 zebra_phase	66
14.3 hpdfdbl_cell Struct Reference	66
14.3.1 Detailed Description	67
14.3.2 Field Documentation	67
14.3.2.1 canvas_cb	67
14.3.2.2 colspan	67
14.3.2.3 content	67
14.3.2.4 content_cb	67
14.3.2.5 content_style	68
14.3.2.6 delta_x	68
14.3.2.7 delta_y	68
14.3.2.8 height	68
14.3.2.9 label	68
14.3.2.10 label_cb	68
14.3.2.11 parent_cell	69
14.3.2.12 rowspan	69
14.3.2.13 style_cb	69
14.3.2.14 textwidth	69
14.3.2.15 width	69
14.4 hpdfdbl_cell_spec Struct Reference	70
14.4.1 Detailed Description	70
14.4.2 Field Documentation	70
14.4.2.1 canvas_cb	70
14.4.2.2 col	71
14.4.2.3 colspan	71
14.4.2.4 content_cb	71
14.4.2.5 label	71
14.4.2.6 label_cb	71
14.4.2.7 row	72
14.4.2.8 rowspan	72
14.4.2.9 style_cb	72
14.5 hpdfdbl_errcode_entry Struct Reference	72
14.5.1 Detailed Description	72
14.5.2 Field Documentation	73

14.5.2.1 errcode	73
14.5.2.2 errstr	73
14.6 hpdftbl_spec Struct Reference	73
14.6.1 Detailed Description	74
14.6.2 Field Documentation	74
14.6.2.1 cell_spec	74
14.6.2.2 cols	74
14.6.2.3 content_cb	74
14.6.2.4 height	74
14.6.2.5 label_cb	75
14.6.2.6 post_cb	75
14.6.2.7 rows	75
14.6.2.8 style_cb	75
14.6.2.9 title	75
14.6.2.10 use_header	76
14.6.2.11 use_labelgrid	76
14.6.2.12 use_labels	76
14.6.2.13 width	76
14.6.2.14 xpos	76
14.6.2.15 ypos	77
14.7 hpdftbl_theme Struct Reference	77
14.7.1 Detailed Description	77
14.7.2 Field Documentation	77
14.7.2.1 bottom_vmargin_factor	78
14.7.2.2 content_style	78
14.7.2.3 header_style	78
14.7.2.4 inner_hborder	78
14.7.2.5 inner_tborder	78
14.7.2.6 inner_vborder	79
14.7.2.7 label_style	79
14.7.2.8 outer_border	79
14.7.2.9 title_style	79
14.7.2.10 use_header_row	79
14.7.2.11 use_label_grid_style	80
14.7.2.12 use_labels	80
14.7.2.13 use_zebra	80
14.7.2.14 zebra_color1	80
14.7.2.15 zebra_color2	80
14.7.2.16 zebra_phase	81
14.8 line_dash_style Struct Reference	81
14.8.1 Detailed Description	81
14.8.2 Field Documentation	81

14.8.2.1 dash_ptn	81
14.8.2.2 num	81
14.9 text_style Struct Reference	82
14.9.1 Detailed Description	82
14.9.2 Field Documentation	82
14.9.2.1 background	82
14.9.2.2 color	83
14.9.2.3 font	83
14.9.2.4 fsize	83
14.9.2.5 halign	83
15 File Documentation	85
15.1 unit_test.inc.h File Reference	85
15.1.1 Detailed Description	86
15.1.2 Macro Definition Documentation	86
15.1.2.1 TUTEX_MAIN	86
15.1.3 Function Documentation	87
15.1.3.1 setup_dummy_content()	87
15.1.3.2 setup_dummy_content_label()	87
15.1.3.3 setup_filename()	88
15.1.3.4 setup_hpdlf()	88
15.1.3.5 stroke_to_file()	89
15.1.4 Variable Documentation	90
15.1.4.1 run_as_unit_test	90
15.2 unit_test.inc.h	90
15.3 bootstrap.sh File Reference	92
15.3.1 Detailed Description	92
15.4 dbgld.sh File Reference	93
15.4.1 Detailed Description	93
15.5 docupload.sh.in File Reference	93
15.5.1 Detailed Description	94
15.5.2 Variable Documentation	94
15.5.2.1 GITHUB_USER	94
15.5.2.2 PDFFILE_COPY	94
15.6 stdbld.sh File Reference	94
15.6.1 Detailed Description	95
15.7 config.h	95
15.8 hpdtbl.c File Reference	96
15.8.1 Detailed Description	99
15.8.2 Function Documentation	100
15.8.2.1 chktbl()	100
15.8.2.2 HPDF_RoundedCornerRectangle()	101

15.8.2.3 <code>hpdftbl_clear_spanning()</code>	101
15.8.2.4 <code>hpdftbl_create()</code>	102
15.8.2.5 <code>hpdftbl_create_title()</code>	102
15.8.2.6 <code>hpdftbl_default_table_error_handler()</code>	103
15.8.2.7 <code>hpdftbl_destroy()</code>	103
15.8.2.8 <code>hpdftbl_encoding_text_out()</code>	104
15.8.2.9 <code>hpdftbl_get_anchor_top_left()</code>	104
15.8.2.10 <code>hpdftbl_get_errstr()</code>	104
15.8.2.11 <code>hpdftbl_get_last_auto_height()</code>	105
15.8.2.12 <code>hpdftbl_get_last_err_file()</code>	105
15.8.2.13 <code>hpdftbl_get_last_errcode()</code>	106
15.8.2.14 <code>hpdftbl_set_anchor_top_left()</code>	106
15.8.2.15 <code>hpdftbl_set_background()</code>	106
15.8.2.16 <code>hpdftbl_set_bottom_vmargin_factor()</code>	107
15.8.2.17 <code>hpdftbl_set_cell()</code>	107
15.8.2.18 <code>hpdftbl_set_cell_content_style()</code>	108
15.8.2.19 <code>hpdftbl_set_cellspan()</code>	109
15.8.2.20 <code>hpdftbl_set_col_content_style()</code>	109
15.8.2.21 <code>hpdftbl_set_colwidth_percent()</code>	110
15.8.2.22 <code>hpdftbl_set_content()</code>	111
15.8.2.23 <code>hpdftbl_set_content_style()</code>	111
15.8.2.24 <code>hpdftbl_set_errhandler()</code>	112
15.8.2.25 <code>hpdftbl_set_header_halign()</code>	113
15.8.2.26 <code>hpdftbl_set_header_style()</code>	113
15.8.2.27 <code>hpdftbl_set_inner_grid_style()</code>	114
15.8.2.28 <code>hpdftbl_set_inner_hgrid_style()</code>	114
15.8.2.29 <code>hpdftbl_set_inner_tgrid_style()</code>	115
15.8.2.30 <code>hpdftbl_set_inner_vgrid_style()</code>	116
15.8.2.31 <code>hpdftbl_set_label_style()</code>	116
15.8.2.32 <code>hpdftbl_set_labels()</code>	117
15.8.2.33 <code>hpdftbl_set_line_dash()</code>	117
15.8.2.34 <code>hpdftbl_set_min_rowheight()</code>	118
15.8.2.35 <code>hpdftbl_set_outer_grid_style()</code>	118
15.8.2.36 <code>hpdftbl_set_row_content_style()</code>	119
15.8.2.37 <code>hpdftbl_set_tag()</code>	120
15.8.2.38 <code>hpdftbl_set_text_encoding()</code>	120
15.8.2.39 <code>hpdftbl_set_title()</code>	121
15.8.2.40 <code>hpdftbl_set_title_halign()</code>	121
15.8.2.41 <code>hpdftbl_set_title_style()</code>	122
15.8.2.42 <code>hpdftbl_set_zebra()</code>	122
15.8.2.43 <code>hpdftbl_set_zebra_color()</code>	123
15.8.2.44 <code>hpdftbl_stroke()</code>	123

15.8.2.45	hpdtbl_stroke_from_data()	124
15.8.2.46	hpdtbl_stroke_pdfdoc()	125
15.8.2.47	hpdtbl_use_header()	125
15.8.2.48	hpdtbl_use_labelgrid()	126
15.8.2.49	hpdtbl_use_labels()	127
15.9	hpdtbl.h File Reference	127
15.9.1	Detailed Description	134
15.9.2	Macro Definition Documentation	135
15.9.2.1	_HPDFTBL_SET_ERR	135
15.9.2.2	_HPDFTBL_SET_ERR_EXTRA	135
15.9.2.3	DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR	137
15.9.2.4	hpdtbl_cm2dpi	137
15.9.3	Typedef Documentation	137
15.9.3.1	hpdf_text_style_t	138
15.9.3.2	hpdtbl_callback_t	138
15.9.3.3	hpdtbl_canvas_callback_t	138
15.9.3.4	hpdtbl_cell_spec_t	138
15.9.3.5	hpdtbl_cell_t	139
15.9.3.6	hpdtbl_content_callback_t	139
15.9.3.7	hpdtbl_content_style_callback_t	139
15.9.3.8	hpdtbl_error_handler_t	139
15.9.3.9	hpdtbl_grid_style_t	140
15.9.3.10	hpdtbl_line_dashstyle_t	140
15.9.3.11	hpdtbl_spec_t	140
15.9.3.12	hpdtbl_t	140
15.9.3.13	hpdtbl_text_align_t	140
15.9.3.14	hpdtbl_theme_t	141
15.9.4	Enumeration Type Documentation	141
15.9.4.1	hpdtbl_dashstyle	141
15.9.4.2	hpdtbl_text_align	141
15.9.5	Function Documentation	143
15.9.5.1	chktbl()	143
15.9.5.2	HPDF_RoundedCornerRectangle()	143
15.9.5.3	hpdtbl_apply_theme()	144
15.9.5.4	hpdtbl_clear_spanning()	144
15.9.5.5	hpdtbl_create()	145
15.9.5.6	hpdtbl_create_title()	145
15.9.5.7	hpdtbl_default_table_error_handler()	146
15.9.5.8	hpdtbl_destroy()	146
15.9.5.9	hpdtbl_destroy_theme()	147
15.9.5.10	hpdtbl_encoding_text_out()	147
15.9.5.11	hpdtbl_get_anchor_top_left()	148

15.9.5.12 <code>hpdfctl_get_default_theme()</code>	148
15.9.5.13 <code>hpdfctl_get_errstr()</code>	148
15.9.5.14 <code>hpdfctl_get_last_auto_height()</code>	149
15.9.5.15 <code>hpdfctl_get_last_err_file()</code>	149
15.9.5.16 <code>hpdfctl_get_last_errcode()</code>	150
15.9.5.17 <code>hpdfctl_hpdf_get_errstr()</code>	150
15.9.5.18 <code>hpdfctl_set_anchor_top_left()</code>	151
15.9.5.19 <code>hpdfctl_set_background()</code>	151
15.9.5.20 <code>hpdfctl_set_bottom_vmargin_factor()</code>	151
15.9.5.21 <code>hpdfctl_set_canvas_cb()</code>	152
15.9.5.22 <code>hpdfctl_set_cell()</code>	152
15.9.5.23 <code>hpdfctl_set_cell_canvas_cb()</code>	153
15.9.5.24 <code>hpdfctl_set_cell_canvas_dyncb()</code>	154
15.9.5.25 <code>hpdfctl_set_cell_content_cb()</code>	154
15.9.5.26 <code>hpdfctl_set_cell_content_dyncb()</code>	155
15.9.5.27 <code>hpdfctl_set_cell_content_style()</code>	156
15.9.5.28 <code>hpdfctl_set_cell_content_style_cb()</code>	156
15.9.5.29 <code>hpdfctl_set_cell_content_style_dyncb()</code>	157
15.9.5.30 <code>hpdfctl_set_cell_label_cb()</code>	158
15.9.5.31 <code>hpdfctl_set_cell_label_dyncb()</code>	158
15.9.5.32 <code>hpdfctl_set_cellspan()</code>	159
15.9.5.33 <code>hpdfctl_set_col_content_style()</code>	159
15.9.5.34 <code>hpdfctl_set_colwidth_percent()</code>	160
15.9.5.35 <code>hpdfctl_set_content()</code>	160
15.9.5.36 <code>hpdfctl_set_content_cb()</code>	161
15.9.5.37 <code>hpdfctl_set_content_dyncb()</code>	162
15.9.5.38 <code>hpdfctl_set_content_style()</code>	163
15.9.5.39 <code>hpdfctl_set_content_style_cb()</code>	163
15.9.5.40 <code>hpdfctl_set_content_style_dyncb()</code>	164
15.9.5.41 <code>hpdfctl_set_dlhandle()</code>	164
15.9.5.42 <code>hpdfctl_set_errhandler()</code>	165
15.9.5.43 <code>hpdfctl_set_header_halign()</code>	165
15.9.5.44 <code>hpdfctl_set_header_style()</code>	166
15.9.5.45 <code>hpdfctl_set_inner_grid_style()</code>	166
15.9.5.46 <code>hpdfctl_set_inner_hgrid_style()</code>	167
15.9.5.47 <code>hpdfctl_set_inner_tgrid_style()</code>	168
15.9.5.48 <code>hpdfctl_set_inner_vgrid_style()</code>	168
15.9.5.49 <code>hpdfctl_set_label_cb()</code>	169
15.9.5.50 <code>hpdfctl_set_label_dyncb()</code>	170
15.9.5.51 <code>hpdfctl_set_label_style()</code>	170
15.9.5.52 <code>hpdfctl_set_labels()</code>	171
15.9.5.53 <code>hpdfctl_set_min_rowheight()</code>	171

15.9.5.54	hpdfctl_set_outer_grid_style()	172
15.9.5.55	hpdfctl_set_row_content_style()	172
15.9.5.56	hpdfctl_set_tag()	173
15.9.5.57	hpdfctl_set_text_encoding()	173
15.9.5.58	hpdfctl_set_title()	174
15.9.5.59	hpdfctl_set_title_halign()	174
15.9.5.60	hpdfctl_set_title_style()	175
15.9.5.61	hpdfctl_set_zebra()	176
15.9.5.62	hpdfctl_set_zebra_color()	176
15.9.5.63	hpdfctl_stroke()	176
15.9.5.64	hpdfctl_stroke_from_data()	177
15.9.5.65	hpdfctl_stroke_grid()	178
15.9.5.66	hpdfctl_stroke_pdfdoc()	178
15.9.5.67	hpdfctl_table_widget_letter_buttons()	179
15.9.5.68	hpdfctl_use_header()	180
15.9.5.69	hpdfctl_use_labelgrid()	180
15.9.5.70	hpdfctl_use_labels()	181
15.9.5.71	hpdfctl_widget_hbar()	181
15.9.5.72	hpdfctl_widget_segment_hbar()	182
15.9.5.73	hpdfctl_widget_slide_button()	183
15.9.5.74	hpdfctl_widget_strength_meter()	183
15.10	hpdfctl.h	185
15.11	hpdfctl_callback.c File Reference	192
15.11.1	Detailed Description	193
15.11.2	Function Documentation	193
15.11.2.1	hpdfctl_set_canvas_cb()	193
15.11.2.2	hpdfctl_set_cell_canvas_cb()	194
15.11.2.3	hpdfctl_set_cell_canvas_dyncb()	195
15.11.2.4	hpdfctl_set_cell_content_cb()	195
15.11.2.5	hpdfctl_set_cell_content_dyncb()	196
15.11.2.6	hpdfctl_set_cell_content_style_cb()	196
15.11.2.7	hpdfctl_set_cell_content_style_dyncb()	197
15.11.2.8	hpdfctl_set_cell_label_cb()	198
15.11.2.9	hpdfctl_set_cell_label_dyncb()	198
15.11.2.10	hpdfctl_set_content_cb()	200
15.11.2.11	hpdfctl_set_content_dyncb()	201
15.11.2.12	hpdfctl_set_content_style_cb()	201
15.11.2.13	hpdfctl_set_content_style_dyncb()	202
15.11.2.14	hpdfctl_set_dlhandle()	202
15.11.2.15	hpdfctl_set_label_cb()	203
15.11.2.16	hpdfctl_set_label_dyncb()	203
15.12	hpdfctl_errstr.c File Reference	204

15.12.1 Detailed Description	204
15.12.2 Function Documentation	204
15.12.2.1 hpdf_tbl_hpdpf_get_errstr()	204
15.13 hpdf_tbl_grid.c File Reference	205
15.13.1 Detailed Description	205
15.13.2 Function Documentation	205
15.13.2.1 hpdf_tbl_stroke_grid()	205
15.14 hpdf_tbl_theme.c File Reference	206
15.14.1 Detailed Description	207
15.14.2 Macro Definition Documentation	207
15.14.2.1 HPDFTBL_DEFAULT_CONTENT_STYLE	207
15.14.2.2 HPDFTBL_DEFAULT_HEADER_STYLE	208
15.14.2.3 HPDFTBL_DEFAULT_INNER_HGRID_STYLE	208
15.14.2.4 HPDFTBL_DEFAULT_INNER_VGRID_STYLE	208
15.14.2.5 HPDFTBL_DEFAULT_LABEL_STYLE	208
15.14.2.6 HPDFTBL_DEFAULT_OUTER_GRID_STYLE	209
15.14.2.7 HPDFTBL_DEFAULT_ZEBRA_COLOR1	209
15.14.2.8 HPDFTBL_DEFAULT_ZEBRA_COLOR2	209
15.14.3 Function Documentation	209
15.14.3.1 hpdf_tbl_apply_theme()	209
15.14.3.2 hpdf_tbl_destroy_theme()	210
15.14.3.3 hpdf_tbl_get_default_theme()	210
15.15 hpdf_tbl_widget.c File Reference	211
15.15.1 Detailed Description	212
15.15.2 Macro Definition Documentation	212
15.15.2.1 FALSE	212
15.15.2.2 TRUE	212
15.15.3 Function Documentation	212
15.15.3.1 hpdf_tbl_table_widget_letter_buttons()	212
15.15.3.2 hpdf_tbl_widget_hbar()	213
15.15.3.3 hpdf_tbl_widget_segment_hbar()	214
15.15.3.4 hpdf_tbl_widget_slide_button()	214
15.15.3.5 hpdf_tbl_widget_strength_meter()	215
16 Example Documentation	217
16.1 example01.c	217
16.2 tut_ex01.c	222
16.3 tut_ex02.c	223
16.4 tut_ex02_1.c	223
16.5 tut_ex03.c	223
16.6 tut_ex04.c	224
16.7 tut_ex05.c	224

16.8 tut_ex06.c	225
16.9 tut_ex07.c	225
16.10 tut_ex08.c	226
16.11 tut_ex09.c	227
16.12 tut_ex10.c	228
16.13 tut_ex11.c	229
16.14 tut_ex12.c	229
16.15 tut_ex13_1.c	229
16.16 tut_ex13_2.c	230
16.17 tut_ex14.c	231
16.18 tut_ex15.c	232
16.19 tut_ex15_1.c	233
16.20 tut_ex20.c	233
16.21 tut_ex30.c	234
Index	235

Chapter 1

Overview

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

For the long version see [Building from source](#)

If the necessary [pre-requisites](#) are fulfilled the distributed tar-ball can be built with:

```
$ tar xzf libhpdfctl-<version>.tar.gz
$ cd libhpdfctl-<version>
$ ./configure && make
```

If any libraries are missing the `configure` process will discover this and tell what needs to be installed.

Note

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

To verify the build run

```
$ make check
```

If everything works you should see a *Success!* message. After checking that the library passes the unit-tests it can be installed with

```
$ make install
```

If successfully this will install the library in the `/usr/local` subtree. It will build and install both a static and dynamic version of the library.

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 build the library:

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 different (and incompatible) versions of `libiconv` readily available for OSX. One library that 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 it is safe to assume that a library called `/usr/lib/iconv.dylib` is available.

However, if one tries to list this library in `/usr/lib` there will not be a `libiconv.dylib`. Still, if the code is linked 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. `clang`) 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* with suffix `*.tbd` for each supported SDK version (tbd stands for "text based description"). They are small text files with 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 is located 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`

If the build complains about `libiconv` the following steps could be taken to try to track down the problem:

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

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

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

2. Find out the SDK path that is actively used

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

3. Check your `PATH` variable

```
$> echo $PATH
```

2.3 Building the library from source

There are two levels of rebuilding the library that we will discuss

1. Using a build environment to rebuild the library (i.e. building from the supplied tar-ball)
2. Rebuilding from a cloned repo and rebuild the build environment from scratch. As a principle no generated files are stored in the repo.

2.3.1 Rebuilding using an existing build environment

Rebuilding the library using a pre-configured build environment requires `gcc` (or `clang`) and `make` together with the standard C/C++ libraries to build the library.

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-x.y.z.tar.gz
$ cd libhpdf-x.y.z
$ ./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. As a shortcut two utility scripts are included that give some extra CFLAGS flags to either compile the library for production use `./scripts/stdbld.sh` or for debugging `./scripts/dbgbld.sh` (See [Some notes on Debugging](#))

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 installed in order to fully rebuild from a cloned repo.

1. A complete set of GNU compiler chain (or on OSX clang)
2. 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 bootstrapping necessary to create 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` top 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 with

```
$ 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: SUCCESS! 20/20 tests passed.
Info: =====
```

To install the library use

```
$> 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>
```

This will run the `gdb` debugger from command line. For debugging from within a IDE (like Netbeans, Clion, etc.) use the static library method.

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 minimize the build time during development. To rebuild the *html* documentation build the target

```
$> make html
```

and to rebuild the *PDF* version build (assuming you have LaTeX installed)

```
$> 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, this is on a best effort basis since I have no longer access to a Windows system to verify the workings.

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 `hpdf_tbl` 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 an infrastructure for the examples

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 basic setup for all our examples. We will create a document in A4 size that have one page that will be written to a file whose name is taken from the program arguments. 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(_hpdf_tbl_jmp_env)) {
        HPDF_Free(pdf_doc);
        return EXIT_FAILURE;
    }
    setup_hpdf(&pdf_doc, &pdf_page, FALSE);
    create_table_<EXAMPLE NAME>(pdf_doc, pdf_page);
    if ( -1 == stroke_to_file(pdf_doc, argc, argv) )
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}
```

In order to make the example code consistent and focused on the table library and not on the general creating of PDF document we will include the supporting Haru set-up code in an include file and instead of the `main()` function shown above we will replace it with a macro with one parameter; the table function to be called to set-up the table example (see [TUTEX_MAIN\(\)](#)).

All our example code will therefore be a fully standalone programs but structured in way not to obscure the actual table creation with a lot of boiler-plate PDF set-up code. All tutorial example programs `tut_ex<nn>` will therefore have the following general structure:

```
#include "unit_test.inc.h"

void
create_table_XXXX(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    ...
}
TUTEX_MAIN(create_table_XXXX, FALSE)
```

The second argument to the [TUTEX_MAIN\(\)](#) macro determines if the example should be generated with gridlines on the paper. This is useful for precisely position the table on a page.

In the `examples` directory the full source code for the setup and stroke functions 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 with one A4 page and the [stroke_to_file\(\)](#) strokes the document to an output file which depends on the program argument.

Note

If any of the test programs are run without any arguments the output file will be stored in the `out` directory and have the same name as the basename of the program with a `"*.pdf"` suffix. If exactly one filename is specified as an argument then this is the file the output will be written to.

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. We will follow the framework outlined above. Our first example is `tut_ex01.c`

First we construct a table handle for a 2x2 table

```
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);
```

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");
```

Here we note that:

- Cells are referred to starting from the top left cell that is cell (0x0).
- The `NULL` argument (4th argument) will be explained shortly.

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 either by specifying an overall table height or a minimum row height using `hpdf_tbl_set_min_rowheight()`)

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. The paper coordinate system has its origin in the lower left corner of the paper. We position the top left of the table *1 cm* below and *1 cm* to the right of the top left corner of the paper. To make this easier we make use of the constant `A4PAGE_HEIGHT_CM` and make the table *5 cm* wide as follows:

```
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
```

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 `hpdftbl_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 and use the macro to create a main function `TUTEX_MAIN()` as follows:

```
hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex01, FALSE)
```

and we are done!

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

```
1
4 #include "unit_test.inc.h"
5
9 void
10 create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
11     const size_t num_rows = 2;
12     const size_t num_cols = 2;
13
14     hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
15
16     hpdftbl_set_cell(tbl, 0, 0, NULL, "Cell 0x0");
17     hpdftbl_set_cell(tbl, 0, 1, NULL, "Cell 0x1");
18     hpdftbl_set_cell(tbl, 1, 0, NULL, "Cell 1x0");
19     hpdftbl_set_cell(tbl, 1, 1, NULL, "Cell 1x1");
20
21     // We have to specify the top left position on the PDF as well as the width.
22     // We let the library automatically determine the height of the table based
23     // on the font and number of rows.
24     HPDF_REAL xpos = hpdftbl_cm2dpi(1);
25     HPDF_REAL ypos = hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM - 1);
26     HPDF_REAL width = hpdftbl_cm2dpi(5);
27     HPDF_REAL height = 0; // Calculate height automatically
28
29     // Stroke the table to the page
30     hpdftbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
31 }
32
33 TUTEX_MAIN(create_table_ex01, FALSE)
```

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 `hpdftbl_stroke_grid()` function such a grid can be displayed on a page to help with positioning.

In our infrastructure set-up this call is controlled by setting the second macro parameter to `TRUE`, i.e. `TUTEX_MAIN(create_table_ex01, FALSE)`

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**.



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. In our setup it is possible to generate the grid by setting the third argument to `setup_hpdpf()` to `TRUE`. This can be done by updating the `TUTEX_MAIN()` macro

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

```
hpdftbl_set_content(hpdftbl_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:

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:

```
create_table_ex02(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, 2, 2);
    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(5);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdftbl_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 `hpdftbl_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 `hpdftbl_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 `hpdftbl_use_header()` function. By modifying the code above and add this line we get the following code and resulting table

```
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_content_with_header(&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);
}
```

The resulting table can be seen in **Figure 4**. We also modified the dummy data to have the work "Header" text for `row==0` 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
Label 5	Label 6
Label 7	Label 8

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	Label 2
Label 3	Label 4
Label 5	Label 6
Label 7	Label 8

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

Note

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

To create dummy data for both content and labels we use the function `setup_dummy_content_label()`

```
void setup_dummy_content_label(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 (strictly speaking the call to `hpdf_tbl_use_labelgrid()` is not necessary since by default the short gridlines will be enabled when labels are first enabled.)

```
setup_dummy_content_label(&content, &labels, num_rows, num_cols);
hpdf_tbl_set_content(tbl, content);
hpdf_tbl_set_labels(tbl, labels);
```

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

```
hpdf_tbl_use_labels(tbl, TRUE);
hpdf_tbl_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.

```
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";
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_content_label(&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);
}
```

[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 `hpdftbl_set_title_style()` and `hpdftbl_set_title_halign()`

The complete code for this example is shown below

```
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";
    hpdftbl_t tbl = hpdftbl_create_title(num_rows, num_cols, table_title);
    content_t content, labels;
    setup_dummy_content_label(&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);
}
```

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 `hpdftbl_set_content_style()`
2. Adjust one entire column using `hpdftbl_set_col_content_style()`
3. Adjust one entire row in using `hpdftbl_set_row_content_style()`
4. Adjust individual cells using `hpdftbl_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

Cell and row spanning

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 column 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 $> 100\%$.

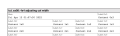


Figure 9: *Adjusting width of first columns. [tut_ex08.c](#) *

Chapter 5

Using 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 library for each cell and returns a string which is used as the data to be displayed. The signature for a cell content callback is defined by the type `hpdf_tbl_content_callback_t` which is a pointer to a function defined as:

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

This signature is also used for label callbacks. For style setting callback the signature is instead defined as

```
typedef
_Bool (*hpdf_tbl_content_style_callback_t)(void *, size_t, size_t, char *content, hpdf_text_style_t *);
```

To understand this lets start defining a callback function to specify content (or a label) that follows this signature.

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

The parameters in the callback are

Parameter	Description
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.
row	The cell row
col	The cell column

It is possible to specify a callback to adjust content, style, and labels. A callback function can be specified to be used for every cell in the table or only for a specific cell. This can also be mixed in order to have, for example, one generic callback for most cells and have a different callback for a specific cell. Any callback set for a cell will override the callback set for the table

The API to specify these callbacks are:

API	Description
<code>hpdftbl_set_content_cb()</code>	Specify a content callback for the entire table.
<code>hpdftbl_set_content_style_cb()</code>	Specify a style callback for the entire table.
<code>hpdftbl_set_label_cb()</code>	Specify a label callback for the entire table.
<code>hpdftbl_set_cell_content_cb()</code>	Specify callback for an individual cell. A cell callback will override a potential table callback.
<code>hpdftbl_set_cell_content_style_cb()</code>	Specify a style callback for an individual cell. A cell callback will override a potential table callback.
<code>hpdftbl_set_canvas_cb()</code>	This is an advanced callback to allow for low level painting directly on the canvas that is the cell area 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. The string pointer returned from a callback is never stored in the table. only printed. It is therefore perfectly possible to have a static allocated buffer in the callback function that is used to construct the content and returned from the callback.

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 date callback functions we need to add a date in the top left corner of the table that reflects the current date and time.

```
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;
}
```

This would be sufficient for normal usage. However, the source code in [tut_ex06.c](#) that illustrates callbacks is slightly different:

```
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
```

```

    }
}

```

The reason for this is that all these examples also serve as unit tests for the library. The way the unit tests work is by comparing the output from all these examples with stored, manually checked "correct" versions of the output. Since any date changes will make the file different we must make the dates a known value whe the examples are run as unit teets. This we know when the flag `run_as_unit_test` is true and in that case a "dummy" static date is used.

The content and label functions can then be written as follows

```

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 %zux%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 the 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:

```

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_content_cb(tbl, cb_content);
    hpdf_tbl_set_label_cb(tbl, cb_labels);
    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
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

```

Running this example gives the result shown in **Figure 7**. below, the full source code can be found in [tut_ex06.c](#)



tut_ex06: 2x2 table with callbacks	
Content 00 x 00	Label 00x00:
Content 01 x 01	Label 01x01:

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

5.3 Dynamic (late binding) callbacks

Warning

This is an advanced concept and while simple in theory it does have some hidden "gotchas".

All callback functions discussed above must exist at compile time so that the address of the functions can be determined by the compiler. As we will discuss later it is possible to define a table as a data structure to avoid having to write several lines of code in defining a table.

Such a data structure could in theory be stored in a database or as a text file. In that case it will not be possible to specify a callback function since the address of function is determined at link time.

Fortunately it is possible to specify a function name (as a string) and have the standard C-library locate where that function is stored and return a pointer to it. This pointer is then the same as if the callback had been bound at compile time.

There is an analog set of functions that takes a string name of the function and looks up the actual function pointer and set that as the callback.

Those analogue functions are

API	Description
hpdftbl_set_dlhandle()	Option to set dynamic lib handle
hpdftbl_set_content_dyncb()	Table content late binding
hpdftbl_set_label_dyncb()	Table label late binding
hpdftbl_set_cell_label_dyncb()	Table cell label latex binding
hpdftbl_set_content_style_dyncb()	Table style late binding
hpdftbl_set_cell_content_style_dyncb()	Table cell content late binding
hpdftbl_set_cell_canvas_dyncb()	Cell canvas callback

They are identical to the already described "ordinary" setting callback functions with the difference these functions take a string as argument rather than a function pointer.

5.3.1 Using late binding

Using late binding is very similar to what we did in the previous examples. We start by defining the callback functions we need

```
char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}

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;
}

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

```
}

```

It is however one crucial detail that cannot be overlooked. **None of the callbacks functions can be static!** If they are static they won't be found

Then it is really simple. We create the table with the function that should now be familiar and then add the callbacks with the names of the callback functions as so

```
hpdftbl_set_content_dyncb(tbl, "cb_content");
hpdftbl_set_label_dyncb(tbl, "cb_labels");
hpdftbl_set_cell_content_dyncb(tbl, 0, 0, "cb_date");
```

In this way it is possible to specify the entire table structure as a text structure that could be stored in a database or as a text file with just the name of the callback functions. However, care must be taken that they are named exactly as they are specified.

The complete table function is shown below and the full example be found in [tut_ex30.c](#)

```
void
create_table_ex30(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex30: Table with dynamic 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_dyncb(tbl, "cb_content");
    hpdftbl_set_label_dyncb(tbl, "cb_labels");
    hpdftbl_set_cell_content_dyncb(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);
}
```


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

Argument	Description
tbl	The table in where the error happened. Note: This might be NULL since not all errors happen within the context of a table
r, c	The row and column if the error happens in a specified cell, otherwise these will be (-1,-1)
err	The internal error code. This si 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(EXIT_FAILURE);  
}
```

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 `hpdfdbl_default_table_error_handler` so to use this trivial error handler just add the following line to your code `hpdfdbl_set_errhandler(hpdfdbl_default_table_error_handler);`

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

6.1 Using emulated exception handling

As can be seen above the default error handler terminates the running process with a call to `exit(EXIT_FAILURE)`. Terminating the process might not always be appropriate (especially for a daemon process). An alternative way to handle a fault state is to use `setjmp()/longjmp()` to simulate an exception handling.

In the program setup code a jump point is established and then if an error is detected the error handler will jump to the set jump point.

For example, all tutorial examples share the same `main()` function as shown below

```
main(int argc, char **argv) {
    HPDF_Doc pdf_doc;
    HPDF_Page pdf_page;
    run_as_unit_test = 2==argc ;
    if (setjmp(_hpdfdbl_jump_env)) {
        return EXIT_FAILURE;
    }
    hpdfdbl_set_errhandler(table_error_handler);
    setup_hpdf(&pdf_doc, &pdf_page, _showgrid_);
    _tbl_(pdf_doc, pdf_page);
    if( -1 == stroke_to_file(pdf_doc, argc, argv) )
        return EXIT_FAILURE;
    else
        return EXIT_SUCCESS;
}
```

The relevant part here is the `setjmp(_hpdfdbl_jump_env)` code which establish a jump destination. A basic error handler that uses this could now look like this:

```
static void
table_error_handler(hpdfdbl_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, hpdfdbl_get_errstr(err), r, c);
    } else {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" \n", err, hpdfdbl_get_errstr(err));
    }
    longjmp(_hpdfdbl_jump_env, 1);
}
```

If an error occur the `longjmp()` will come to the `setjmp()` point and since it returns the value of 1 it will enter the if-statement free the doc structure and then terminate the process by exiting the `main()` function.

In a more complex program it might be useful to instead of exiting the process give the user an error message, do any cleanup (such as freeing the PDF document) and try again if this perhaps was a recoverable error.

The actual error handler used in the tutorial examples is slightly longer as it prints all available information from the error handling "subsystem" such as which file and line number (in the library) where the error was triggered and any optional extra information was given in regard to the error mode. In addition, a stacktrace is also generated to `stderr`

The real (production grade) error handler therefore looks as shown below

```
table_error_handler(hpdfdbl_t t, int r, int c, int err) {
    int lineno;
    char *filename;
    char *extrainfo;
    hpdfdbl_get_last_err_file(&lineno, &filename, &extrainfo);
    if (r > -1 && c > -1) {
        fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)", err,
            hpdfdbl_get_errstr(err), r, c);
    } else {
        fprintf(stderr, "*** Table Error: [%d] \"%s\"", err,
            hpdfdbl_get_errstr(err));
    }
}
```

```

if( filename != NULL ) {
    fprintf(stderr, " in %s:%d", filename, lineno);
}
if( extrainfo != NULL ) {
    fprintf(stderr, ". Info: \"%s\"\\n", extrainfo);
}
else {
    fprintf(stderr, "\\n");
}
// Also print the available stacktrace
void* callstack[128];
int i, frames = backtrace(callstack, 128);
char** callstack_sym = backtrace_symbols(callstack, frames);
if( callstack_sym != NULL ) {
    fprintf(stderr, "Stacktrace:\\n");
    for (i = 0; i < frames; ++i) {
        fprintf(stderr, "%s\\n", callstack_sym[i]);
    }
    free(callstack_sym);
}
longjmp(_hpdf_tbl_jmp_env, 1);
}

```

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
```

6.2 Additional information

When an error is triggered the file name and line number in the library where the error was triggered is saved as well as an optional information string that some error states might set.

All this extra information can be retrieved by the library function [hpdf_tbl_get_last_err_file\(\)](#)

Note

The file name and line number displayed is always the point in the library that discovered the error state. It does not indicate the file name and line number of the client code that triggered the error as the error is discovered in the library routines.

6.3 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 *
hpdf_tbl_hpdf_get_errstr(const HPDF_STATUS err_code)
```

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

6.4 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, 30);
    hpdftbl_set_colwidth_percent(tbl, 1, 30);
    content_t content;
    setup_dummy_content(&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](#).

If we simulate a "typo" and add a deliberate error by making the column widths larger than 100% by writing

```
hpdftbl_set_colwidth_percent(tbl, 0, 80);
hpdftbl_set_colwidth_percent(tbl, 1, 30);
```

When this code is then executed the following will be printed to standard error and the process will be stopped.

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

Chapter 7

Font and style 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_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 }
```

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

```
...
hpdftbl_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 tables give colors as an integer triple there is exists a macro to make this conversion easier

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

which will allow the easier specification of color such as

```
#define HPDF_COLOR_ORANGE      HPDF_RGB_CONVERT(0xF5, 0xD0, 0x98);
#define HPDF_COLOR_ALMOST_BLACK HPDF_RGB_CONVERT(0x14, 0x14, 0x14);
```

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
hpdftbl_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 0x1	Header 0x2	Header 0x3
Extra long Header 1x1	Content 0x1	Content 0x2	Content 0x3
Extra long Header 2x1	Content 0x1	Content 0x2	Content 0x3
Extra long Header 3x1	Content 0x1	Content 0x2	Content 0x3

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 [hpdtbl_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 :

API	Description
int hpdtbl_apply_theme (hpdtbl_t t, hpdtbl_theme_t *theme)	Apply the given theme to a table
hpdtbl_theme_t * hpdtbl_get_default_theme (void)	Get the default theme into a new allocated structure
int hpdtbl_destroy_theme (hpdtbl_theme_t *theme)	Free the memory used by a theme

Note

It is the responsibility of the user of the library to destroy the theme structure by ensuring that `hpdf_tbl_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
hpdf_tbl_set_inner_tgrid_style(hpdf_tbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color, hpdf_tbl_line_dashstyle_t dashstyle);
int
hpdf_tbl_set_inner_vgrid_style(hpdf_tbl_t t,
                               HPDF_REAL width, HPDF_RGBColor color, hpdf_tbl_line_dashstyle_t dashstyle);
int
```

```

hpdf_tbl_set_inner_hgrid_style(hpdf_tbl_t t,
                               HPDF_REAL width, HPDF_RGBColor color, hpdf_tbl_line_dashstyle_t dashstyle);
int
hpdf_tbl_set_inner_grid_style(hpdf_tbl_t t,
                              HPDF_REAL width, HPDF_RGBColor color, hpdf_tbl_line_dashstyle_t dashstyle);

```

Each type of gridlines can be adjusted with line width, color and style. The last function in the list, [hpdf_tbl_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 [hpdf_tbl_dashstyle](#) and grid style functions [hpdf_tbl_set_inner_grid_style\(\)](#), [hpdf_tbl_set_inner_vgrid_style\(\)](#) and [hpdf_tbl_set_inner_hgrid_style\(\)](#)

Dash Style	Illustration
LINE_SOLID	
LINE_DOT1	
LINE_DOT2	
LINE_DOT3	
LINE_DOT4	
LINE_DASH1	
LINE_DASH2	
LINE_DASH3	
LINE_DASH4	
LINE_DASH5	
LINE_DASHDOT1	
LINE_DASHDOT2	

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_content(&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
hpdftbl_set_zebra(hpdftbl_t t, _Bool use, int phase);
int
hpdftbl_set_zebra_color(hpdftbl_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 0 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;
    hpdftbl_t tbl = hpdftbl_create(num_rows, num_cols);
    content_t content;
    setup_dummy_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_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_content(&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

Note

Another way to hide a gridline is to set its width to 0.

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 (almost, we'll get back to the limitations) 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 structure 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 [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 (see section [Using callbacks](#))

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 hpdftbl_cell_spec {
    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;
} hpdftbl_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

```
hpdftbl_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=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
    .width=hpdftbl_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 compared to the table creation code we have seen in the previous examples and consists of only one line of code

```
void
create_table_ex13_1(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    hpdftbl_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 (H10)	Header (H10)	Header (H10)
Content (F100)	Content (F100)	Content (F100)
Content (F100)	Content (F100)	Content (F100)
Content (F100)	Content (F100)	Content (F100)

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 data (and how it is created).

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

```
hpdtbl_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},
    HPDTBL_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 `HPDTBL_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.

```
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=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
    .width=hpdftbl_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 16**

Mark Ericson	12 Sep 2021	123 Downer Mews	London
NW2 HB3	mark.p.ericson@myfinemail.com	+44734 354 184 56	+44771 938 137 11

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.

Mark Ericson	12 Sep 2021	123 Downer Mews	London
NW2 HB3	mark.p.ericson@myfinemail.com	+44734 354 184 56	+44771 938 137 11

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.

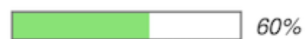
9.1.1 1. Segmented horizontal bar example

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 `hpdfdbl_set_canvas_cb()` and `hpdfdbl_set_cell_canvas_cb()` functions. The callback function itself has to follow the canvas callback signature which is defined as

```
typedef void (*hpdfdbl_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 it's 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)
{ ... }
...
hpdfdbl_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 has slightly different defining functions depending on what they display and are defined as follows.

9.2.1 Segmented horizontal bar defining function

```
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)
```

9.2.2 Horizontal bar defining function

```
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)
```

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

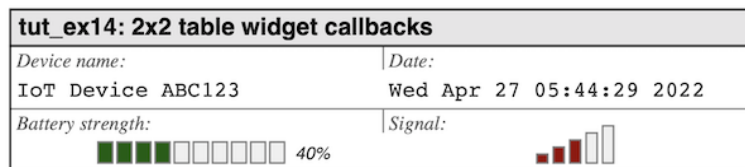


Figure 9.1 tut_ex14.c

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

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 `hpdtbl_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

- `hpdtbl_set_cell()` *Set content text in specified cell.*
- `hpdtbl_set_tag()` *Set the table tag. The tag is a `void *` and can be anything. The tag is the first parameter of all callbacks.*
- `hpdtbl_set_title()` *Set title text of table.*
- `hpdtbl_set_labels()` *Set label texts for the table from 1D-data array.*
- `hpdtbl_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.

- `hpdtbl_set_content_cb()` *Set table content callback.*
- `hpdtbl_set_cell_content_cb()` *Set cell content callback.*
- `hpdtbl_set_cell_content_style_cb()` *Set the cell style callback.*
- `hpdtbl_set_content_style_cb()` *Set the table style callback.*
- `hpdtbl_set_label_cb()` *Set table label callback.*
- `hpdtbl_set_cell_label_cb()` *Set the cell label callback.*
- `hpdtbl_set_canvas_cb()` *Set table canvas callback.*
- `hpdtbl_set_cell_canvas_cb()` *Set the cell canvas callback.*

10.8 Dynamic (late binding) callback handling

These are callbacks which set a function at runtime to be used as callback. This is useful when specifying the table for example as a structure stored in a database or in a file. The callback function is then specified as a string (the name of the callback function which is then resolved at runtime).

- `hpdtbl_set_dlhandle()`
 *Set the dynamic library load handle as returned by `dlopen()` or one of the predefined handles. By default, the handle is set to the predefined handle `RTLD_DEFAULT`. See `man dlsym`. This handle will control how the search for the name of the function will be conducted. The default will find any functions defined in any images linked and any libraries linked at compile time. It will **not** find functions defined in libraries that are dynamically loaded. In that case you should specify the handle returned by `dlopen()`.

- [hpdftbl_set_content_dyncb\(\)](#) *Set the name for the table content callback.*
- [hpdftbl_set_cell_content_dyncb\(\)](#) *Set the name for the cell content callback.*
- [hpdftbl_set_label_dyncb\(\)](#)
- [hpdftbl_set_cell_label_dyncb\(\)](#) *Set the name for the cell label content callback.*
- [hpdftbl_set_content_style_dyncb\(\)](#) *Set the name for the table content style callback.*
- [hpdftbl_set_cell_content_style_dyncb\(\)](#) *Set the name for the cell content style callback.*
- [hpdftbl_set_cell_canvas_dyncb\(\)](#) *Set the name for the cell canvas callback.*

10.9 Text encoding

- [hpdftbl_set_text_encoding\(\)](#) *Specify text encodation to use.*
- [hpdftbl_encoding_text_out\(\)](#) *Stroke a text with current encoding.*

10.10 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:

grid_style	Specification for table grid lines	57
hpdftbl	Core table handle	58
hpdftbl_cell	Specification of individual cells in the table	66
hpdftbl_cell_spec	Used in data driven table creation	70
hpdftbl_errcode_entry	An entry in the error string table	72
hpdftbl_spec	Used in data driven table creation	73
hpdftbl_theme	Define a set of styles into a table theme	77
line_dash_style	Definition of a dashed line style	81
text_style	Specification of a text style	82

Chapter 13

File Index

13.1 File List

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

unit_test.inc.h	Common functions for all unit-test/examples	85
bootstrap.sh	Bootstrap the autotools environment and configure a build setup	92
dbgblld.sh	Setup a build environment for debugging	93
docupload.sh.in	Upload the generated documentation to the github pages doc site for the author	93
stdbld.sh	Setup a build environment for production build	94
config.h	95
hpdftbl.c	Main module for flexible table drawing with HPDF library	96
hpdftbl.h	Header file for libhpdftbl	127
hpdftbl_callback.c	Routines for callback function for flexible table drawing with HPDF library	192
hpdftbl_errstr.c	Utility module to translate HPDF error codes to human readable strings	204
hpdftbl_grid.c	Create a grid on a document for positioning	205
hpdftbl_theme.c	Functions for theme handling	206
hpdftbl_widget.c	Support for drawing widgets	211

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](#), [tut_ex20.c](#), and [tut_ex30.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

`hpdtbl_canvas_callback_t canvas_cb`

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

Referenced by [hpdtbl_set_canvas_cb\(\)](#).

14.2.2.3 cells

`hpdtbl_cell_t* cells`

Reference to all an array of cells in the table

Referenced by [hpdtbl_clear_spanning\(\)](#), [hpdtbl_create_title\(\)](#), [hpdtbl_set_cell\(\)](#), [hpdtbl_set_cell_canvas_cb\(\)](#), [hpdtbl_set_cell_content_cb\(\)](#), [hpdtbl_set_cell_content_style\(\)](#), [hpdtbl_set_cell_content_style_cb\(\)](#), [hpdtbl_set_cell_label_cb\(\)](#), [hpdtbl_set_cellspan\(\)](#), [hpdtbl_set_content\(\)](#), and [hpdtbl_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 [hpdtbl_create_title\(\)](#), and [hpdtbl_set_colwidth_percent\(\)](#).

14.2.2.5 cols

`size_t cols`

Number of columns in table

Referenced by [hpdtbl_clear_spanning\(\)](#), [hpdtbl_create_title\(\)](#), [hpdtbl_destroy\(\)](#), [hpdtbl_set_cellspan\(\)](#), [hpdtbl_set_colwidth_percent\(\)](#), [hpdtbl_set_content\(\)](#), [hpdtbl_set_labels\(\)](#), and [hpdtbl_set_row_content_style\(\)](#).

14.2.2.6 content_cb

`hpdtbl_content_callback_t content_cb`

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

Referenced by [hpdtbl_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_cellspan\(\)](#), [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 [hpdftbl_set_tag\(\)](#).

14.2.2.24 title_style

`hpdf_text_style_t title_style`

Title style

Referenced by [hpdftbl_set_title_halign\(\)](#), and [hpdftbl_set_title_style\(\)](#).

14.2.2.25 title_txt

`char* title_txt`

Title text

Referenced by [hpdftbl_create_title\(\)](#), [hpdftbl_destroy\(\)](#), and [hpdftbl_set_title\(\)](#).

14.2.2.26 use_cell_labels

`_Bool use_cell_labels`

Flag to determine if cell labels should be used

Referenced by [hpdftbl_apply_theme\(\)](#), and [hpdftbl_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 [hpdftbl_apply_theme\(\)](#), and [hpdftbl_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

`hpdfctl_canvas_callback_t canvas_cb`

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

Referenced by [hpdfctl_set_cell_canvas_cb\(\)](#).

14.3.2.2 colspan

`size_t colspan`

Number of column this cell spans

Referenced by [hpdfctl_clear_spanning\(\)](#), [hpdfctl_set_cell\(\)](#), and [hpdfctl_set_cellspan\(\)](#).

14.3.2.3 content

`char* content`

String reference for cell content

Referenced by [hpdfctl_set_cell\(\)](#), and [hpdfctl_set_content\(\)](#).

14.3.2.4 content_cb

`hpdfctl_content_callback_t content_cb`

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

Referenced by [hpdfctl_set_cell_content_cb\(\)](#).

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

Referenced by [hpdftbl_set_cell_content_style\(\)](#).

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_cell\(\)](#), and [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

Referenced by [hpdftbl_set_cell_label_cb\(\)](#).

14.3.2.11 parent_cell

```
struct hpdfdbl_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 [hpdfdbl_clear_spanning\(\)](#), [hpdfdbl_set_cell\(\)](#), [hpdfdbl_set_cell_canvas_cb\(\)](#), [hpdfdbl_set_cell_content_cb\(\)](#), [hpdfdbl_set_cell_content_style_cb\(\)](#), [hpdfdbl_set_cell_label_cb\(\)](#), and [hpdfdbl_set_cellspan\(\)](#).

14.3.2.12 rowspan

```
size_t rowspan
```

Number of rows this cell spans

Referenced by [hpdfdbl_clear_spanning\(\)](#), [hpdfdbl_set_cell\(\)](#), and [hpdfdbl_set_cellspan\(\)](#).

14.3.2.13 style_cb

```
hpdfdbl_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

Referenced by [hpdfdbl_set_cell_content_style_cb\(\)](#).

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:

- [hpdfdbl.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 `hpdftbl_spec_t` structure. The array should have one entry for each cell in the table.

See also

[hpdftbl_stroke_from_data\(\)](#)

Examples

[example01.c](#), and [tut_ex13_2.c](#).

14.4.2 Field Documentation

14.4.2.1 canvas_cb

`hpdftbl_canvas_callback_t canvas_cb`

Canvas callback for this cell

Referenced by [hpdftbl_stroke_from_data\(\)](#).

14.4.2.2 col

`size_t col`

Row for specified cell

Referenced by [hpdfdbl_stroke_from_data\(\)](#).

14.4.2.3 colspan

`unsigned colspan`

Number of columns the specified cell should span

Referenced by [hpdfdbl_stroke_from_data\(\)](#).

14.4.2.4 content_cb

[hpdfdbl_content_callback_t](#) `content_cb`

Content callback for this cell

Referenced by [hpdfdbl_stroke_from_data\(\)](#).

14.4.2.5 label

`char* label`

The label for this cell

Referenced by [hpdfdbl_stroke_from_data\(\)](#).

14.4.2.6 label_cb

[hpdfdbl_content_callback_t](#) `label_cb`

Label callback for this cell

Referenced by [hpdfdbl_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:

- [hpdfctl_errstr.c](#)

14.6 hpdfctl_spec Struct Reference

Used in data driven table creation.

```
#include <hpdfctl.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](#)
- [hpdfctl_content_callback_t](#) [content_cb](#)
- [hpdfctl_content_callback_t](#) [label_cb](#)
- [hpdfctl_content_style_callback_t](#) [style_cb](#)
- [hpdfctl_callback_t](#) [post_cb](#)
- [hpdfctl_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

`hpdfctl_grid_style_t` inner_vborder

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

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_get_default_theme\(\)](#).

14.7.2.7 label_style

`hpdf_text_style_t` label_style

Label text style

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_get_default_theme\(\)](#).

14.7.2.8 outer_border

`hpdfctl_grid_style_t` outer_border

Table outer border style

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_get_default_theme\(\)](#).

14.7.2.9 title_style

`hpdf_text_style_t` title_style

Table title text style

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_get_default_theme\(\)](#).

14.7.2.10 use_header_row

`_Bool` use_header_row

Flag if header row should be used

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_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 [hpdfctl_apply_theme\(\)](#), and [hpdfctl_get_default_theme\(\)](#).

14.7.2.12 use_labels

`_Bool use_labels`

Flag if cell labels should be used

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_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:

- [hpdfctl.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 [hpdftbl_apply_theme\(\)](#), [hpdftbl_set_background\(\)](#), [hpdftbl_set_cell_content_style\(\)](#), [hpdftbl_set_content_style\(\)](#), [hpdftbl_set_header_style\(\)](#), [hpdftbl_set_label_style\(\)](#), and [hpdftbl_set_title_style\(\)](#).

14.9.2.2 color

HPDF_RGBColor color

Font color

Examples

[tut_ex09.c](#).

Referenced by [hpdfctl_apply_theme\(\)](#), [hpdfctl_set_cell_content_style\(\)](#), [hpdfctl_set_content_style\(\)](#), [hpdfctl_set_header_style\(\)](#), [hpdfctl_set_label_style\(\)](#), and [hpdfctl_set_title_style\(\)](#).

14.9.2.3 font

char* font

Font face name

Examples

[tut_ex09.c](#).

Referenced by [hpdfctl_apply_theme\(\)](#), [hpdfctl_set_cell_content_style\(\)](#), [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_cell_content_style\(\)](#), [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 unit_test.inc.h File Reference

Common functions for all unit-test/examples.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <execinfo.h>
#include <unistd.h>
#include <libgen.h>
#include <sys/stat.h>
#include <sys/utsname.h>
#include <hpdf.h>
#include <math.h>
#include <setjmp.h>
#include <time.h>
#include "../src/hpdftbl.h"
```

Macros

- `#define TUTEX_MAIN(_tbl_, _showgrid_)`

Macro to create a `main()` function to call the table creation function for each example. The name to the table function is given as the first and only argument to the macro.

Typedefs

- `typedef char ** content_t`

An array of char pointers.

Functions

- void `setup_hpdf` (HPDF_Doc *pdf_doc, HPDF_Page *pdf_page, _Bool addgrid)
Create a new PDF document with one page in A4 format.
- char * `setup_filename` (int argc, char **argv)
Return a pointer to a static buffer that holds the filename to be used for the PDF page.
- int `stroke_to_file` (HPDF_Doc pdf_doc, int argc, char **argv)
Stroke the created PDF page to a file.
- void `setup_dummy_content` (content_t *content, size_t rows, size_t cols)
Create an array of char pointers to simulate real table data.
- void `setup_dummy_content_label` (content_t *content, content_t *labels, size_t rows, size_t cols)
Create both array of char pointers to simulate real table content as well as an array of simulated labels.

Variables

- _Bool `run_as_unit_test` = FALSE
For the case when we use this example as a unit/integration test we do not want data such as dates, times, and system-information to be updated since the checks compare the result to a stored copy of the PDF file.
- jmp_buf `_hpdftbl_jmp_env`

15.1.1 Detailed Description

Common functions for all unit-test/examples.

15.1.2 Macro Definition Documentation

15.1.2.1 TUTEX_MAIN

```
#define TUTEX_MAIN(
    _tbl_,
    _showgrid_ )
```

Value:

```
int \
main(int argc, char **argv) { \
    HPDF_Doc pdf_doc; \
    HPDF_Page pdf_page; \
    run_as_unit_test = 2==argc ; \
    if (setjmp(_hpdftbl_jmp_env)) { \
        return EXIT_FAILURE; \
    } \
    hpdftbl_set_errhandler(table_error_handler); \
    setup_hpdf(&pdf_doc, &pdf_page, _showgrid_); \
    _tbl_(pdf_doc, pdf_page); \
    if( -1 == stroke_to_file(pdf_doc, argc, argv) ) \
        return EXIT_FAILURE; \
    else \
        return EXIT_SUCCESS; \
}
```

Macro to create a main() function to call the table creation function for each example. The name to the table function is given as the first and only argument to the macro.

Parameters

<code>_tbl_</code>	The name of the main table creation function to be called.
<code>_↔ showgrid↔ _</code>	Set to TRUE to display a dot-grid on the paper, FALSE otherwise.

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](#), [tut_ex20.c](#), and [tut_ex30.c](#).

15.1.3 Function Documentation

15.1.3.1 `setup_dummy_content()`

```
void setup_dummy_content (
    content_t * content,
    size_t rows,
    size_t cols )
```

Create an array of char pointers to simulate real table data.

Parameters

out	<i>content</i>	A pointer to an array of char pointers
in	<i>rows</i>	Number of rows in table
in	<i>cols</i>	Number of columns in table

Examples

[tut_ex02.c](#), [tut_ex10.c](#), [tut_ex11.c](#), [tut_ex12.c](#), [tut_ex15.c](#), and [tut_ex15_1.c](#).

15.1.3.2 `setup_dummy_content_label()`

```
void setup_dummy_content_label (
    content_t * content,
    content_t * labels,
    size_t rows,
    size_t cols )
```

Create both array of char pointers to simulate real table content as well as an array of simulated labels.

Parameters

out	<i>content</i>	A pointer to an array of char pointers to represent content in a table
out	<i>labels</i>	A pointer to an array of char pointers to represent labels in a table
in	<i>rows</i>	Number of rows in table
in	<i>cols</i>	Number of columns in table

Examples

[tut_ex04.c](#), [tut_ex05.c](#), and [tut_ex20.c](#).

15.1.3.3 setup_filename()

```
char * setup_filename (
    int argc,
    char ** argv )
```

Return a pointer to a static buffer that holds the filename to be used for the PDF page.

The filename is create from the program arguments. If there are no arguments the file name will have the same basename as the program (but with an added `".pdf"` suffix and path `"out/"`).

If there are exactly on argument this will in its entirety (incl path) be taken as the full name for the file.

Warning

It is the calling functions responsibility to check for a NULL return value.

More than one program argument will generate an error (return NULL)

Parameters

<i>argc</i>	The main() functions argc argument
<i>argv</i>	The main() functions argv argument

Returns

NULL on error, otherwise a pointer to a static string that holds the filename.

Referenced by [stroke_to_file\(\)](#).

15.1.3.4 setup_hpdf()

```
void setup_hpdf (
    HPDF_Doc * pdf_doc,
```

```
HPDF_Page * pdf_page,
_Bool addgrid )
```

Create a new PDF document with one page in A4 format.

This initializes a basic PDF document object and a page object that can then be used. A typical calling sequence would be:

```
HPDF_Doc pdf_doc;
HPDF_Page pdf_page;
setup_hpdf(&pdf_doc, &pdf_page, FALSE);
```

The pdf_doc and pdf_page can subsequently be used in the other PDF functions.

By setting the addgrid to TRUE the paper background will be gridlines with coordinate system units in points. This is very useful to precisely position text and graphics on a page.

Parameters

out	<i>pdf_doc</i>	A pointer The document handle
out	<i>pdf_page</i>	A pointer to a page handle
in	<i>addgrid</i>	Set to TRUE to add coordinate grid lines to the paper (in points)

15.1.3.5 stroke_to_file()

```
int stroke_to_file (
    HPDF_Doc pdf_doc,
    int argc,
    char ** argv )
```

Stroke the created PDF page to a file.

The filename to be used is determined by the [setup_filename\(\)](#) function

Parameters

<i>pdf_doc</i>	The PDF document to be written
<i>argc</i>	The main() functions argc argument
<i>argv</i>	The main() functions argv argument

Returns

0 on success, -1 on failure

See also

[setup_filename\(\)](#), [hpdfbl_stroke_pdfdoc\(\)](#)

Examples

[example01.c](#).

15.1.4 Variable Documentation

15.1.4.1 run_as_unit_test

```
_Bool run_as_unit_test = FALSE
```

For the case when we use this example as a unit/integration test we do not want data such as dates, times, and system-information to be updated since the checks compare the result to a stored copy of the PDF file.

When this is true it is used as a flag in, for example, the call-back functions to suppress the printing of dates and times so the result may be compared to the stored versions of the document and not be dependent on a specific time and date.

Examples

[example01.c](#), [tut_ex06.c](#), [tut_ex07.c](#), [tut_ex08.c](#), [tut_ex14.c](#), and [tut_ex30.c](#).

15.2 unit_test.inc.h

[Go to the documentation of this file.](#)

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <execinfo.h>
6
7 #if !(defined _WIN32 || defined __WIN32__)
8 #include <unistd.h>
9 #include <libgen.h>
10 #include <sys/stat.h>
11 #include <sys/utsname.h>
12 #endif
13
14 #include <hpdf.h>
15 #include <math.h>
16 #include <setjmp.h>
17 #include <time.h>
18 #include "../src/hpdftbl.h"
19
20 _Bool run_as_unit_test = FALSE;
21
22 // For simulated exception handling
23 jmp_buf _hpdftbl_jmp_env;
24
25 #ifndef _MSC_VER
26 // Silent gcc about unused "arg" in the callback and error functions
27 #pragma GCC diagnostic push
28 #pragma GCC diagnostic ignored "-Wunused-parameter"
29 #pragma GCC diagnostic ignored "-Wunused-function"
30 #endif
31
32 static void
33 error_handler(HPDF_STATUS error_no, HPDF_STATUS detail_no,
34              void *user_data) {
35     fprintf(stderr, "*** PDF ERROR: \"%s\", [0x%04X : %d]\n",
36             hpdftbl_hpdf_get_errstr(error_no), (unsigned int)error_no, (int)detail_no);
37     longjmp(_hpdftbl_jmp_env, 1);
38 }
39
40 static void
41 table_error_handler(hpdftbl_t t, int r, int c, int err) {
42     int lineno;
43     char *filename;
44     char *extrainfo;
45
46     hpdftbl_get_last_err_file(&lineno, &filename, &extrainfo);
47     if (r > -1 && c > -1) {
48         fprintf(stderr, "*** Table Error: [%d] \"%s\" at cell (%d, %d)", err,

```



```

80         hpdfdbl_get_errstr(err), r, c);
81     } else {
82         fprintf(stderr, "*** Table Error: [%d] \"%s\"", err,
83             hpdfdbl_get_errstr(err));
84     }
85     if( filename != NULL ) {
86         fprintf(stderr, " in %s:%d", filename, lineno);
87     }
88     if( extrainfo != NULL ) {
89         fprintf(stderr, ". Info: \"%s\"", extrainfo);
90     }
91     else {
92         fprintf(stderr, "\n");
93     }
94
95     // Also print the available stacktrace
96     void* callstack[128];
97     int i, frames = backtrace(callstack, 128);
98     char** callstack_sym = backtrace_symbols(callstack, frames);
99     if( callstack_sym != NULL ) {
100         fprintf(stderr, "Stacktrace:\n");
101         for (i = 0; i < frames; ++i) {
102             fprintf(stderr, "%s\n", callstack_sym[i]);
103         }
104         free(callstack_sym);
105     }
106
107     longjmp(_hpdfdbl_jmp_env, 1);
108 }
109
110
111 #ifndef _MSC_VER
112 #pragma GCC diagnostic pop
113 #endif
114
115 // Setup a new PDF document with one page
116 void
117 setup_hpdf(HPDF_Doc* pdf_doc, HPDF_Page* pdf_page, _Bool addgrid) {
118     *pdf_doc = HPDF_New(error_handler, NULL);
119     *pdf_page = HPDF_AddPage(*pdf_doc);
120     HPDF_SetCompressionMode(*pdf_doc, HPDF_COMP_ALL);
121     HPDF_Page_SetSize(*pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
122     if (addgrid) {
123         hpdfdbl_stroke_grid(*pdf_doc, *pdf_page);
124     }
125 }
126
127 char *
128 setup_filename(int argc, char **argv) {
129     static char file[1024];
130     if ( 2==argc ) {
131         strncpy(file, argv[1], sizeof file);
132         file[sizeof(file)-1] = 0;
133     } else if ( 1==argc ) {
134         char fbuffer[255];
135         strncpy(fbuffer, argv[0], sizeof fbuffer);
136         fbuffer[sizeof(fbuffer) - 1] = 0;
137         char *bname = basename(fbuffer);
138         snprintf(file, sizeof file, "out/%s.pdf", bname);
139     } else {
140         return NULL;
141     }
142     return file;
143 }
144
145 int
146 stroke_to_file(HPDF_Doc pdf_doc, int argc, char **argv) {
147     char *file;
148     if( NULL == (file=setup_filename(argc, argv)) ) {
149         fprintf(stderr, "ERROR: Unknown arguments!\n");
150         return -1;
151     }
152
153     printf("Sending to file \"%s\" ...\n", file);
154     if ( -1 == hpdfdbl_stroke_pdfdoc(pdf_doc, file) ) {
155         fprintf(stderr, "ERROR: Cannot save to file. Does the full directory path exist?\n");
156         return -1;
157     }
158     printf("Done.\n");
159     return 0;
160 }
161
162
163 typedef char **content_t;
164
165 void setup_dummy_content(content_t *content, size_t rows, size_t cols) {
166     char buff[255];

```

```

228     *content = calloc(rows*cols, sizeof(char*));
229     size_t cnt = 0;
230     for (size_t r = 0; r < rows; r++) {
231         for (size_t c = 0; c < cols; c++) {
232             snprintf(buff, sizeof(buff), "Content %zu", cnt);
233             (*content)[cnt] = strdup(buff);
234             cnt++;
235         }
236     }
237 }
238
239 void setup_dummy_content_label(content_t *content, content_t *labels, size_t rows, size_t cols) {
240     char buff[255];
241     *content = calloc(rows*cols, sizeof(char*));
242     *labels = calloc(rows*cols, sizeof(char*));
243     size_t cnt = 0;
244     for (size_t r = 0; r < rows; r++) {
245         for (size_t c = 0; c < cols; c++) {
246             snprintf(buff, sizeof(buff), "Content %zu", cnt);
247             (*content)[cnt] = strdup(buff);
248             snprintf(buff, sizeof(buff), "Label %zu", cnt);
249             (*labels)[cnt] = strdup(buff);
250             cnt++;
251         }
252     }
253 }
254
255 #define TUTEX_MAIN(_tbl_, _showgrid_) int \
256 main(int argc, char **argv) { \
257     HPDF_Doc pdf_doc; \
258     HPDF_Page pdf_page; \
259     run_as_unit_test = 2==argc ; \
260     if (setjmp(_hpdf_tbl_jmp_env)) { \
261         return EXIT_FAILURE; \
262     } \
263     hpdf_tbl_set_errhandler(table_error_handler); \
264     setup_hpdf(&pdf_doc, &pdf_page, _showgrid_); \
265     _tbl_(pdf_doc, pdf_page); \
266     if( -1 == stroke_to_file(pdf_doc, argc, argv) ) \
267         return EXIT_FAILURE; \
268     else \
269         return EXIT_SUCCESS; \
270 }

```

15.3 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.

15.3.1 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

15.4 dbgblid.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

15.4.1 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:

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

-q : Quiet

-h : Print help and exit

See LICENSE file. (C) 2022 Johan Persson johan162@gmail.com

15.5 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

15.5.1 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

15.5.2 Variable Documentation

15.5.2.1 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.

15.5.2.2 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.

15.6 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

15.6.1 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

15.7 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 1.3.0"

```

```
63
64 /* Define to the one symbol short name of this package. */
65 #define PACKAGE_TARNAME "libhpdftbl"
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 "1.3.0"
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 "1.3.0"
```

15.8 hpdftbl.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 <libgen.h>
#include <sys/stat.h>
#include <setjmp.h>
#include "hpdftbl.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.
- void [hpdftbl_get_last_err_file](#) (int *lineno, char **file, char **extrainfo)
Get the filename and line number where the last error occurred.
- [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.
- _Bool [chktbl](#) (hpdftbl_t t, size_t r, size_t c)
Internal function. Check that a row and column are within the table.
- 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_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_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.

- int `hpdftbl_stroke_pdfdoc` (HPDF_Doc pdf_doc, char *file)

Stroke PDF document to file with check that the directory in path exists.

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.

- int `hpdftbl_err_lineno` = 0

Hold the line number of the last error occurred.

- char * `hpdftbl_err_file` = NULL

Hold the file name where the last error occurred.

- char `hpdftbl_err_extrainfo` [1024] = {0}

Extra info that may be specified at the point of error.

- `hpdftbl_error_handler_t` `hpdftbl_err_handler` = NULL

This stores a pointer to the function acting as the error handler callback.

15.8.1 Detailed Description

Main module for flexible table drawing with HPDF library.

Author

Johan Persson (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.

15.8.2 Function Documentation

15.8.2.1 chktbl()

```
_Bool chktbl (
    hpdftbl_t t,
    size_t r,
    size_t c )
```

Internal function. Check that a row and column are within the table.

Internal function. Check that a row and column are within the table

Parameters

<i>t</i>	Table handle
<i>r</i>	Row
<i>c</i>	Column

Returns

TRUE if within bounds, FALSE otherwise

Referenced by [hpdftbl_set_cell\(\)](#), [hpdftbl_set_cell_canvas_cb\(\)](#), [hpdftbl_set_cell_content_cb\(\)](#), [hpdftbl_set_cell_content_style\(\)](#), [hpdftbl_set_cell_content_style_cb\(\)](#), [hpdftbl_set_cell_label_cb\(\)](#), and [hpdftbl_set_cellspan\(\)](#).

15.8.2.2 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 [hpdftbl_widget_slide_button\(\)](#).

15.8.2.3 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\(\)](#)

15.8.2.4 `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

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](#).

15.8.2.5 `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

Examples

[example01.c](#), [tut_ex05.c](#), [tut_ex06.c](#), [tut_ex07.c](#), [tut_ex08.c](#), [tut_ex14.c](#), and [tut_ex30.c](#).

Referenced by [hpdftbl_create\(\)](#), and [hpdftbl_stroke_from_data\(\)](#).

15.8.2.6 hpdfctl_default_table_error_handler()

```
void hpdfctl_default_table_error_handler (
    hpdfctl_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

[hpdfctl_set_errhandler\(\)](#)

Examples

[tut_ex10.c](#), [tut_ex11.c](#), and [tut_ex12.c](#).

15.8.2.7 hpdfctl_destroy()

```
int hpdfctl_destroy (
    hpdfctl_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 [hpdfctl_stroke_from_data\(\)](#).

15.8.2.8 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

15.8.2.9 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

15.8.2.10 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\(\)](#).

15.8.2.11 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

15.8.2.12 hpdftbl_get_last_err_file()

```
void hpdftbl_get_last_err_file (
    int * lineno,
    char ** file,
    char ** extrainfo )
```

Get the filename and line number where the last error occurred.

Parameters

<i>lineno</i>	Set to the line number where the error occurred
<i>file</i>	Set to the file where the error occurred
<i>extrainfo</i>	Extra info string that may be set at the point of error

15.8.2.13 `hpdfdbl_get_last_errcode()`

```
int hpdfdbl_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](#).

15.8.2.14 `hpdfdbl_set_anchor_top_left()`

```
void hpdfdbl_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
---------------	--

15.8.2.15 `hpdfdbl_set_background()`

```
int hpdfdbl_set_background (
    hpdfdbl_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

15.8.2.16 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 [hpdfctl_apply_theme\(\)](#).

15.8.2.17 hpdfctl_set_cell()

```
int hpdfctl_set_cell (
    hpdfctl_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\(\)](#).

15.8.2.18 hpdfctl_set_cell_content_style()

```
int hpdfctl_set_cell_content_style (
    hpdfctl_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

[hpdfctl_set_content_style\(\)](#)
[hpdfctl_set_cell_content_style_cb\(\)](#)

Examples

[example01.c](#).

Referenced by [hpdfctl_set_col_content_style\(\)](#), and [hpdfctl_set_row_content_style\(\)](#).

15.8.2.19 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\(\)](#).

15.8.2.20 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

[hpdfctl_set_content_style\(\)](#)

[hpdfctl_set_cell_content_style_cb\(\)](#)

15.8.2.21 hpdfctl_set_colwidth_percent()

```
int hpdfctl_set_colwidth_percent (
    hpdfctl_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](#).

15.8.2.22 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](#).

15.8.2.23 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

[hpdfctl_set_cell_content_style\(\)](#)
[hpdfctl_set_cell_content_style_cb\(\)](#)

Examples

[example01.c](#).

Referenced by [hpdfctl_apply_theme\(\)](#).

15.8.2.24 hpdfctl_set_errhandler()

```
hpdfctl_error_handler_t hpdfctl_set_errhandler (  
    hpdfctl_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

[hpdfctl_default_table_error_handler\(\)](#)

Examples

[tut_ex10.c](#), [tut_ex11.c](#), and [tut_ex12.c](#).

15.8.2.25 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\(\)](#).

15.8.2.26 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

[hpdftbl_use_header\(\)](#)

Referenced by [hpdftbl_apply_theme\(\)](#).

15.8.2.27 hpdftbl_set_inner_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.

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

[hpdftbl_set_inner_hgrid_style\(\)](#), [hpdftbl_set_inner_vgrid_style\(\)](#), [hpdftbl_set_outer_grid_style\(\)](#)

15.8.2.28 hpdftbl_set_inner_hgrid_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.

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\(\)](#)

Examples

[tut_ex15_1.c](#), and [tut_ex20.c](#).

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_set_inner_grid_style\(\)](#).

15.8.2.29 hpdfctl_set_inner_tgrid_style()

```
int hpdfctl_set_inner_tgrid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_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

[hpdfctl_set_inner_hgrid_style\(\)](#)

Examples

[tut_ex15_1.c](#), and [tut_ex20.c](#).

Referenced by [hpdfctl_apply_theme\(\)](#).

15.8.2.30 `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\(\)](#).

15.8.2.31 `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\(\)](#).

15.8.2.32 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\(\)](#)

Examples

[example01.c](#), [tut_ex04.c](#), [tut_ex05.c](#), and [tut_ex20.c](#).

15.8.2.33 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](#)

15.8.2.34 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

Examples

[example01.c](#).

15.8.2.35 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\(\)](#)

Examples

[tut_ex20.c](#).

Referenced by [hpdftbl_apply_theme\(\)](#).

15.8.2.36 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

[hpdfdbl_set_content_style\(\)](#)

[hpdfdbl_set_cell_content_style_cb\(\)](#)

15.8.2.37 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

15.8.2.38 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.

15.8.2.39 hpdfctl_set_title()

```
int hpdfctl_set_title (
    hpdfctl_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

[hpdfctl_set_title_style\(\)](#)

[hpdfctl_set_title_halign\(\)](#)

15.8.2.40 hpdfctl_set_title_halign()

```
int hpdfctl_set_title_halign (
    hpdfctl_t t,
    hpdfctl_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

[hpdfctl_set_title\(\)](#)

[hpdfctl_set_title_style\(\)](#)

Examples

[example01.c](#).

Referenced by [hpdftbl_apply_theme\(\)](#).

15.8.2.41 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\(\)](#).

15.8.2.42 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\(\)](#).

15.8.2.43 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\(\)](#).

15.8.2.44 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 hpdftbl_get_last_auto_height()

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](#), [tut_ex20.c](#), and [tut_ex30.c](#).

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.8.2.45 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

[hpdfctl_stroke\(\)](#)

Examples

[example01.c](#), [tut_ex13_1.c](#), and [tut_ex13_2.c](#).

15.8.2.46 hpdfctl_stroke_pdfdoc()

```
int hpdfctl_stroke_pdfdoc (
    HPDF_Doc pdf_doc,
    char * file )
```

Stroke PDF document to file with check that the directory in path exists.

Note: It is a checked error if the full path is longer than 1014 characters

Parameters

<i>pdf_doc</i>	Haru PDF document handle
<i>file</i>	Full pathname of file to write to

Returns

0 on success, -1 on failure

Referenced by [stroke_to_file\(\)](#).

15.8.2.47 hpdfctl_use_header()

```
int hpdfctl_use_header (
    hpdfctl_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\(\)](#).

15.8.2.48 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

[hpdftbl_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](#), [tut_ex20.c](#), and [tut_ex30.c](#).

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.8.2.49 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

[hpdftbl_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](#), [tut_ex20.c](#), and [tut_ex30.c](#).

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.9 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))`
Return the maximum value of numeric variables.
- `#define min(a, b) (((a)<(b)) ? (a):(b))`
Return the minimum value of numeric variables.
- `#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 HPDF_RGB_CONVERT(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 HPDF_RGB_CONVERT(0xF5, 0xD0, 0x98);`
- `#define HPDF_COLOR_ALMOST_BLACK HPDF_RGB_CONVERT(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 LETTERRPAGE_HEIGHT_CM 27.9`
US Letter Height in cm.
- `#define LETTERRPAGE_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;hpdftbl_err_lineno=__LINE__;hpdftbl_err_file=__FILE__; 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_SET_ERR_EXTRA(info) do {strncpy(hpdftbl_err_extrinfo,info,1023);hpdftbl_err_extrinfo[1023]=0;} while(0)`
Set optional extra info at error state. (Currently only used by the late binding setting callback functions)
- `#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 [hpdftbl_text_align](#) [hpdftbl_text_align_t](#)
Enumeration for horizontal text alignment.
- typedef struct [text_style](#) [hpdf_text_style_t](#)
Specification of a text style.
- typedef char *(* [hpdftbl_content_callback_t](#)) (void *, size_t, size_t)
Type specification for the table content callback.
- 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.
- typedef _Bool(* [hpdftbl_content_style_callback_t](#)) (void *, size_t, size_t, char *content, [hpdf_text_style_t](#) *)
Type specification for the content style.
- typedef enum [hpdftbl_dashstyle](#) [hpdftbl_line_dashstyle_t](#)
Possible line dash styles for grid lines.
- typedef struct [grid_style](#) [hpdftbl_grid_style_t](#)
Specification for table grid lines.
- typedef struct [hpdftbl_cell](#) [hpdftbl_cell_t](#)
Type definition for the cell structure.
- typedef struct [hpdftbl](#) * [hpdftbl_t](#)
Table handle is a pointer to the hpdftbl structure.
- typedef void(* [hpdftbl_callback_t](#)) ([hpdftbl_t](#))
Callback type for optional post processing when constructing table from a data array.
- typedef struct [hpdftbl_cell_spec](#) [hpdftbl_cell_spec_t](#)
Used in data driven table creation.
- typedef struct [hpdftbl_spec](#) [hpdftbl_spec_t](#)
Used in data driven table creation.
- typedef struct [hpdftbl_theme](#) [hpdftbl_theme_t](#)
Define a set of styles into a table theme.
- typedef void(* [hpdftbl_error_handler_t](#)) ([hpdftbl_t](#), int, int, int)
TYPe for error handler function.

Enumerations

- enum [hpdftbl_text_align](#) { [LEFT](#) = 0 , [CENTER](#) = 1 , [RIGHT](#) = 2 }
Enumeration for horizontal text alignment.
- enum [hpdftbl_dashstyle](#) {
[LINE_SOLID](#) , [LINE_DOT1](#) , [LINE_DOT2](#) , [LINE_DOT3](#) ,
[LINE_DOT4](#) , [LINE_DASH1](#) , [LINE_DASH2](#) , [LINE_DASH3](#) ,
[LINE_DASH4](#) , [LINE_DASH5](#) , [LINE_DASHDOT1](#) , [LINE_DASHDOT2](#) }
Possible line dash styles for grid lines.

Functions

- [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_stroke](#) (HPDF_Doc pdf, HPDF_Page page, [hpdftbl_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_hpdp_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_get_last_err_file](#) (int *lineno, char **file, char **extrainfo)
- Get the filename and line number where the last error occurred.*

 - 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 [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_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_grid_style](#) ([hpdftbl_t](#) t, HPDF_REAL width, HPDF_RGBColor color, [hpdftbl_line_dashstyle_t](#) dashstyle)
Set inner border 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.
- 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_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_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_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_content_style](#) ([hpdftbl_t](#) t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↔ RGBColor background)
Set style for text content.
- 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_title_style](#) ([hpdftbl_t](#) t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_↔ RGBColor background)
Set the table title style.
- int [hpdftbl_set_cell](#) ([hpdftbl_t](#) t, int r, int c, char *label, char *content)
Set content for specific cell.
- int [hpdftbl_set_tag](#) ([hpdftbl_t](#) t, void *tag)
Set an optional tag for the table.
- 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_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_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_label_cb](#) (hpdftbl_t t, hpdftbl_content_callback_t cb)
Set table label 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_canvas_cb](#) (hpdftbl_t t, hpdftbl_canvas_callback_t cb)
Set cell canvas 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_content_style_cb](#) (hpdftbl_t t, hpdftbl_content_style_callback_t cb)
Set callback to specify cell content style.
- 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.
- void [hpdftbl_set_dlhandle](#) (void *)
Set the handle for scope of dynamic function search.
- int [hpdftbl_set_content_dyncb](#) (hpdftbl_t, char *)
Specify dynamic (late) loading callback content function.
- int [hpdftbl_set_cell_content_dyncb](#) (hpdftbl_t, size_t, size_t, char *)
Specify dynamic (late) loading callback cell content function.
- int [hpdftbl_set_label_dyncb](#) (hpdftbl_t, char *)
Specify dynamic (late) loading callback for table label function.
- int [hpdftbl_set_cell_label_dyncb](#) (hpdftbl_t, size_t, size_t, char *)
Specify dynamic (late) loading callback for cell label function.
- int [hpdftbl_set_content_style_dyncb](#) (hpdftbl_t, char *)
Specify dynamic (late) loading callback for table style function.
- int [hpdftbl_set_cell_content_style_dyncb](#) (hpdftbl_t, size_t, size_t, char *)
Specify dynamic (late) loading callback for cell style function.
- int [hpdftbl_set_cell_canvas_dyncb](#) (hpdftbl_t, size_t, size_t, char *)
Specify dynamic (late) loading callback cell canvas function.
- 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)
Stroke 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_stroke_grid](#) (HPDF_Doc pdf, HPDF_Page page)
- void [hpdftbl_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 [hpdftbl_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 [hpdftbl_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.

- int [hpdfctl_stroke_pdfdoc](#) (HPDF_Doc pdf_doc, char *file)

Stroke PDF document to file with check that the directory in path exists.

- _Bool [chkctl](#) ([hpdfctl_t](#), size_t, size_t)

Internal function. Check that a row and column are within the table.

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.

- int **hpdfctl_err_lineno**

Hold the line number of the last error occurred.

- char * **hpdfctl_err_file**

Hold the file name where the last error occurred.

- char **hpdfctl_err_extrainfo** []

Extra info that may be specified at the point of error.

- [hpdfctl_error_handler_t](#) **hpdfctl_err_handler**

This stores a pointer to the function acting as the error handler callback.

15.9.1 Detailed Description

Header file for libhpdfctl.

Author

Johan Persson (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.

15.9.2 Macro Definition Documentation

15.9.2.1 _HPDFTBL_SET_ERR

```
#define _HPDFTBL_SET_ERR(  
    t,  
    err,  
    r,  
    c ) do {hpdtbl_err_code=err;hpdtbl_err_row=r;hpdtbl_err_col=c;hpdtbl_err_lineno=↵  
__LINE__;hpdtbl_err_file=__FILE__; if(hpdtbl_err_handler) {hpdtbl_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

15.9.2.2 _HPDFTBL_SET_ERR_EXTRA

```
#define _HPDFTBL_SET_ERR_EXTRA(  
    info ) do {strncpy(hpdtbl_err_extrainfo,info,1023);hpdtbl_err_extrainfo[1023]=0;}  
while(0)
```

Set optional extra info at error state. (Currently only used by the late binding setting callback functions)

Parameters

<i>info</i>	Extra info that can be set by a function at a state of error
-------------	--

See also

[hpdfctl_set_label_dynclb\(\)](#), [hpdfctl_set_content_dynclb\(\)](#)

15.9.2.3 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\(\)](#)

15.9.2.4 hpdfctl_cm2dpi

```
#define hpdfctl_cm2dpi(  
    c ) (( (HPDF_REAL) (c) ) / 2.54 * 72)
```

Convert cm to dots using the default resolution (72 DPI)

Parameters

<i>c</i>	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](#), [tut_ex20.c](#), and [tut_ex30.c](#).

15.9.3 Typedef Documentation

15.9.3.1 `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)

15.9.3.2 `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\(\)`](#)

15.9.3.3 `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\(\)`](#)

15.9.3.4 `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\(\)`](#)

15.9.3.5 hpdfctl_cell_t

```
typedef struct hpdfctl_cell hpdfctl_cell_t
```

Type definition for the cell structure.

This is an internal structure that represents an individual cell in the table.

15.9.3.6 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\(\)](#)

15.9.3.7 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\(\)](#)

15.9.3.8 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\(\)](#)

15.9.3.9 `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

15.9.3.10 `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.

15.9.3.11 `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.

15.9.3.12 `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\(\)](#)

15.9.3.13 `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](#)

15.9.3.14 hpdfctl_theme_t

```
typedef struct hpdfctl_theme hpdfctl_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

15.9.4 Enumeration Type Documentation

15.9.4.1 hpdfctl_dashstyle

```
enum hpdfctl_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_"
LINE_DOT4	Dotted line variant 3 "x_x_x_x_"
LINE_DASH1	Dashed line variant 1 "xx_xx_xx_"
LINE_DASH2	Dashed line variant 2 "xx_xx_xx_"
LINE_DASH3	Dashed line variant 3 "xxxx_xxxx_xxxx_"
LINE_DASH4	Dashed line variant 4 "xxxx_xxxx_xxxx_"
LINE_DASH5	Dashed line variant 4 "xxxxxx_xxxxxx_xxxxxx_"
LINE_DASHDOT1	Dashed-dot line variant 1 "xxxxx_xx_xxxx_xx_xxxx_xx_"
LINE_DASHDOT2	Dashed-dot line variant 1 "xxxxxxx_xxx_xxxxxxxx_xxx_xxxxxxxx_xxx_"

15.9.4.2 hpdfctl_text_align

```
enum hpdfctl_text_align
```

Enumeration for horizontal text alignment.

See also[hpdfctl_set_header_halign\(\)](#)[hpdfctl_set_title_halign\(\)](#)[hpdfctl_text_align](#)

Enumerator

LEFT	Left test alignment
CENTER	Center test alignment
RIGHT	Right test alignment

15.9.5 Function Documentation

15.9.5.1 chktbl()

```
_Bool chktbl (
    hpdftbl_t t,
    size_t r,
    size_t c )
```

Internal function. Check that a row and column are within the table.

Internal function. Check that a row and column are within the table

Parameters

<i>t</i>	Table handle
<i>r</i>	Row
<i>c</i>	Column

Returns

TRUE if within bounds, FALSE otherwise

Referenced by [hpdftbl_set_cell\(\)](#), [hpdftbl_set_cell_canvas_cb\(\)](#), [hpdftbl_set_cell_content_cb\(\)](#), [hpdftbl_set_cell_content_style\(\)](#), [hpdftbl_set_cell_content_style_cb\(\)](#), [hpdftbl_set_cell_label_cb\(\)](#), and [hpdftbl_set_cellspan\(\)](#).

15.9.5.2 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 [hpdftbl_widget_slide_button\(\)](#).

15.9.5.3 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\(\)](#).

15.9.5.4 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

[hpdfctl_set_cellspan\(\)](#)

15.9.5.5 hpdfctl_create()

```
hpdfctl_t hpdfctl_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

15.9.5.6 hpdfctl_create_title()

```
hpdfctl_t hpdfctl_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\(\)](#).

15.9.5.7 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\(\)](#)

15.9.5.8 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 [hpdfctl_stroke_from_data\(\)](#).

15.9.5.9 hpdfctl_destroy_theme()

```
int hpdfctl_destroy_theme (
    hpdfctl_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 [hpdfctl_create_title\(\)](#).

15.9.5.10 hpdfctl_encoding_text_out()

```
int hpdfctl_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

15.9.5.11 `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

15.9.5.12 `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\(\)](#).

15.9.5.13 `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\(\)](#).

15.9.5.14 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

15.9.5.15 hpdftbl_get_last_err_file()

```
void hpdftbl_get_last_err_file (
    int * lineno,
    char ** file,
    char ** extrainfo )
```

Get the filename and line number where the last error occurred.

Parameters

<i>lineno</i>	Set to the line number where the error occurred
<i>file</i>	Set to the file where the error occurred
<i>extrainfo</i>	Extra info string that may be set at the point of error

15.9.5.16 `hpdf_tbl_get_last_errcode()`

```
int hpdf_tbl_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

15.9.5.17 `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

<i>err_code</i>	The error code
-----------------	----------------

Returns

A pointer to an error string, NULL if the error code is invalid

See also

[hpdf_tbl_get_errstr\(\)](#)

15.9.5.18 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
---------------	--

15.9.5.19 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

15.9.5.20 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\(\)](#).

15.9.5.21 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\(\)](#)

15.9.5.22 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\(\)](#).

15.9.5.23 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

[hpdftbl_canvas_callback_t](#)
[hpdftbl_set_canvas_cb\(\)](#)

Examples

[example01.c](#), and [tut_ex14.c](#).

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.9.5.24 `hpdfdbl_set_cell_canvas_dyncb()`

```
int hpdfdbl_set_cell_canvas_dyncb (
    hpdfdbl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback cell canvas function.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the `hpdfdbl_get_last_err_file()` to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as canvas callback. This function must follow the signature of a callback function as specified in <code>hpdfdbl_canvas_callback_t</code>

Returns

See also

[hpdfdbl_set_cell_canvas_cb\(\)](#), [hpdfdbl_canvas_callback_t](#)

15.9.5.25 `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

[hpdfctl_set_content_cb\(\)](#)

Examples

[tut_ex06.c](#), [tut_ex07.c](#), [tut_ex08.c](#), and [tut_ex14.c](#).

Referenced by [hpdfctl_stroke_from_data\(\)](#).

15.9.5.26 hpdfctl_set_cell_content_dyncb()

```
int hpdfctl_set_cell_content_dyncb (
    hpdfctl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback cell content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdfctl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in <code>hpdfctl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdfctl_set_content_cb\(\)](#), [hpdfctl_content_callback_t](#)

Examples

[tut_ex30.c](#).

15.9.5.27 `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\(\)](#).

15.9.5.28 `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

[hpdfctl_set_content_style_cb\(\)](#)

Referenced by [hpdfctl_stroke_from_data\(\)](#).

15.9.5.29 hpdfctl_set_cell_content_style_dyncb()

```
int hpdfctl_set_cell_content_style_dyncb (
    hpdfctl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback for cell style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdfctl_content_style_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdfctl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdfctl_content_style_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdfctl_set_cell_content_style_cb\(\)](#), [hpdfctl_content_style_callback_t](#)

15.9.5.30 `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\(\)](#).

15.9.5.31 `hpdftbl_set_cell_label_dyncb()`

```
int hpdftbl_set_cell_label_dyncb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback for cell label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdftbl_content_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdftbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_cell_label_cb\(\)](#), [hpdftbl_content_callback_t](#)

15.9.5.32 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\(\)](#)

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.9.5.33 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

[hpdfctl_set_content_style\(\)](#)

[hpdfctl_set_cell_content_style_cb\(\)](#)

15.9.5.34 hpdfctl_set_colwidth_percent()

```
int hpdfctl_set_colwidth_percent (
    hpdfctl_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

15.9.5.35 hpdfctl_set_content()

```
int hpdfctl_set_content (
    hpdfctl_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()

15.9.5.36 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\(\)](#).

15.9.5.37 hpdftbl_set_content_dyncb()

```
int hpdftbl_set_content_dyncb (
    hpdftbl_t t,
    char * cb_name )
```

Specify dynamic (late) loading callback content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>cb_name</i>	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in <code>hpdftbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_content_cb\(\)](#), [hpdftbl_content_callback_t](#)

Examples

[tut_ex30.c](#).

15.9.5.38 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\(\)](#).

15.9.5.39 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

[hpdfdbl_set_cell_content_style_cb\(\)](#)

Examples

[tut_ex09.c](#).

Referenced by [hpdfdbl_stroke_from_data\(\)](#).

15.9.5.40 hpdfdbl_set_content_style_dyncb()

```
int hpdfdbl_set_content_style_dyncb (
    hpdfdbl_t t,
    char * cb_name )
```

Specify dynamic (late) loading callback for table style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdfdbl_content_style_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdfdbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdfdbl_content_style_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdfdbl_set_content_style_cb\(\)](#), [hpdfdbl_content_style_callback_t](#)

15.9.5.41 hpdfdbl_set_dlhandle()

```
void hpdfdbl_set_dlhandle (
    void * handle )
```

Set the handle for scope of dynamic function search.

When using late binding (some `os_dynb()` functions) the scope for where the runtime searches for the functions can be specified as is discussed in `man 3 dlsym`. By default the library uses `dl_handle` which make the library first searches the current image and then all images it was built against.

If the dynamic callbacks are located in a runtime loaded library then the handle returned by `dlopen()` must be specified as the function will not be found otherwise.

Parameters

<i>handle</i>	Predefined values or the handle returned by <code>dlopen()</code> (see <code>man dlopen</code>)
---------------	--

15.9.5.42 hpdfctl_set_errhandler()

```
hpdfctl_error_handler_t hpdfctl_set_errhandler (
    hpdfctl_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

[hpdfctl_default_table_error_handler\(\)](#)

15.9.5.43 hpdfctl_set_header_halign()

```
int hpdfctl_set_header_halign (
    hpdfctl_t t,
    hpdfctl_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\(\)](#).

15.9.5.44 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

[hpdftbl_use_header\(\)](#)

Referenced by [hpdftbl_apply_theme\(\)](#).

15.9.5.45 hpdftbl_set_inner_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.

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\(\)](#)

15.9.5.46 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\(\)](#).

15.9.5.47 `hpdfctl_set_inner_tgrid_style()`

```
int hpdfctl_set_inner_tgrid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_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

[hpdfctl_set_inner_hgrid_style\(\)](#)

Referenced by [hpdfctl_apply_theme\(\)](#).

15.9.5.48 `hpdfctl_set_inner_vgrid_style()`

```
int hpdfctl_set_inner_vgrid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_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

[hpdfctl_set_inner_grid_style\(\)](#), [hpdfctl_set_inner_hgrid_style\(\)](#)

Referenced by [hpdfctl_apply_theme\(\)](#), and [hpdfctl_set_inner_grid_style\(\)](#).

15.9.5.49 hpdfctl_set_label_cb()

```
int hpdfctl_set_label_cb (
    hpdfctl_t t,
    hpdfctl_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

[hpdfctl_content_callback_t](#)
[hpdfctl_set_cell_label_cb\(\)](#)

Examples

[tut_ex06.c](#), [tut_ex07.c](#), [tut_ex08.c](#), and [tut_ex14.c](#).

Referenced by [hpdfctl_stroke_from_data\(\)](#).

15.9.5.50 `hpdfdbl_set_label_dyncb()`

```
int hpdfdbl_set_label_dyncb (
    hpdfdbl_t t,
    char * cb_name )
```

Specify dynamic (late) loading callback for table label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdfdbl_content_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the `hpdfdbl_get_last_err_file()` to read it back.

Parameters

<i>t</i>	Table handle
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdfdbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdfdbl_set_label_cb\(\)](#), [hpdfdbl_content_callback_t](#)

Examples

[tut_ex30.c](#).

15.9.5.51 `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 [hpdftbl_apply_theme\(\)](#).

15.9.5.52 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\(\)](#)

15.9.5.53 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 thos 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

15.9.5.54 `hpdfctl_set_outer_grid_style()`

```
int hpdfctl_set_outer_grid_style (
    hpdfctl_t t,
    HPDF_REAL width,
    HPDF_RGBColor color,
    hpdfctl_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

[hpdfctl_set_inner_grid_style\(\)](#)

Referenced by [hpdfctl_apply_theme\(\)](#).

15.9.5.55 `hpdfctl_set_row_content_style()`

```
int hpdfctl_set_row_content_style (
    hpdfctl_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\(\)](#)

15.9.5.56 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

15.9.5.57 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.

15.9.5.58 `hpdf_tbl_set_title()`

```
int hpdf_tbl_set_title (
    hpdf_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

[hpdf_tbl_set_title_style\(\)](#)

[hpdf_tbl_set_title_halign\(\)](#)

15.9.5.59 `hpdf_tbl_set_title_halign()`

```
int hpdf_tbl_set_title_halign (
    hpdf_t t,
    hpdf_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

[hpdftbl_set_title\(\)](#)

[hpdftbl_set_title_style\(\)](#)

Referenced by [hpdftbl_apply_theme\(\)](#).

15.9.5.60 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\(\)](#)

Referenced by [hpdftbl_apply_theme\(\)](#).

15.9.5.61 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\(\)](#).

15.9.5.62 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\(\)](#).

15.9.5.63 hpdftbl_stroke()

```
int hpdftbl_stroke (
    HPDF_Doc pdf,
```

```

const HPDF_Page page,
hpdf_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 `hpdf_t_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 hpdf_t_get_last_auto_height()

Returns

-1 on error, 0 if successful

See also

[hpdf_t_get_last_auto_height\(\)](#)
[hpdf_t_stroke_from_data\(\)](#)

Referenced by [hpdf_t_stroke_from_data\(\)](#).

15.9.5.64 hpdftbl_stroke_from_data()

```

int hpdftbl_stroke_from_data (
    HPDF_Doc pdf_doc,
    HPDF_Page pdf_page,
    hpdf_spec_t * tbl_spec,
    hpdf_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

[hpdfdbl_stroke\(\)](#)

15.9.5.65 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

15.9.5.66 hpdfdbl_stroke_pdfdoc()

```
int hpdfdbl_stroke_pdfdoc (
    HPDF_Doc pdf_doc,
    char * file )
```

Stroke PDF document to file with check that the directory in path exists.

Note: It is a checked error if the full path is longer than 1014 characters

Parameters

<i>pdf_doc</i>	Haru PDF document handle
<i>file</i>	Full pathname of file to write to

Returns

0 on success, -1 on failure

Referenced by [stroke_to_file\(\)](#).

15.9.5.67 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](#).

15.9.5.68 `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\(\)](#)

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.9.5.69 `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

[hpdftbl_use_labels\(\)](#)

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.9.5.70 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

[hpdftbl_use_labelgrid\(\)](#)

Referenced by [hpdftbl_stroke_from_data\(\)](#).

15.9.5.71 hpdftbl_widget_hbar()

```
void hpdftbl_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](#).

15.9.5.72 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

Examples

[example01.c](#), and [tut_ex14.c](#).

15.9.5.73 hpdftbl_widget_slide_button()

```
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 )
```

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-öosition 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](#).

15.9.5.74 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](#).

15.10 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
47 #define TRUE 1
48 #endif
49
50 #ifndef FALSE
52 #define FALSE 0
53 #endif
54
55 #ifndef max
56
57 #define max(a,b) (((a)>(b)) ? (a):(b))
58
59 #define min(a,b) (((a)<(b)) ? (a):(b))
60 #endif
61
62 extern int hpdftbl_err_code ;
63
64 extern int hpdftbl_err_row ;
65
66 extern int hpdftbl_err_col ;
67
68 extern int hpdftbl_err_lineno;
69
70 extern char *hpdftbl_err_file;
71
72 extern char hpdftbl_err_extrainfo[];
73
74
75 #define HPDF_FF_TIMES "Times-Roman"
76 #define HPDF_FF_TIMES_ITALIC "Times-Italic"
77 #define HPDF_FF_TIMES_BOLD "Times-Bold"
78 #define HPDF_FF_TIMES_BOLDITALIC "Times-BoldItalic"
79 #define HPDF_FF_HELVETICA "Helvetica"
80 #define HPDF_FF_HELVETICA_ITALIC "Helvetica-Oblique"
81 #define HPDF_FF_HELVETICA_BOLD "Helvetica-Bold"
82 #define HPDF_FF_HELVETICA_BOLDITALIC "Helvetica-BoldOblique"
83 #define HPDF_FF_COURIER "Courier"

```

```

90 #define HPDF_FF_COURIER_BOLD "Courier-Bold"
91 #define HPDF_FF_COURIER_ITALIC "Courier-Oblique"
92 #define HPDF_FF_COURIER_BOLDITALIC "Courier-BoldOblique"
93
94
95
96 #ifdef __cplusplus
100 #define _TO_HPDPF_RGB(r, g, b) \
101     { r / 255.0f, g / 255.0f, b / 255.0f }
102 #else
106 #define HPDF_RGB_CONVERT(r, g, b) \
107     (HPDF_RGBColor) { r / 255.0f, g / 255.0f, b / 255.0f }
108 #endif
109
110 #ifdef __cplusplus
111
112 #define HPDF_COLOR_DARK_RED          { 0.6f, 0.0f, 0.0f }
113 #define HPDF_COLOR_RED               { 1.0f, 0.0f, 0.0f }
114 #define HPDF_COLOR_LIGHT_GREEN      { 0.9f, 1.0f, 0.9f }
115 #define HPDF_COLOR_GREEN             { 0.4f, 0.9f, 0.4f }
116 #define HPDF_COLOR_DARK_GREEN       { 0.05f, 0.37f, 0.02f }
117 #define HPDF_COLOR_DARK_GRAY        { 0.2f, 0.2f, 0.2f }
118 #define HPDF_COLOR_LIGHT_GRAY       { 0.9f, 0.9f, 0.9f }
119 #define HPDF_COLOR_XLIGHT_GRAY      { 0.95f, 0.95f, 0.95f }
120 #define HPDF_COLOR_GRAY             { 0.5f, 0.5f, 0.5f }
121 #define HPDF_COLOR_SILVER            { 0.75f, 0.75f, 0.75f }
122 #define HPDF_COLOR_LIGHT_BLUE       { 1.0f, 1.0f, 0.9f }
123 #define HPDF_COLOR_BLUE             { 0.0f, 0.0f, 1.0f }
124 #define HPDF_COLOR_DARK_BLUE        { 0.0f, 0.0f, 0.6f }
125 #define HPDF_COLOR_WHITE            { 1.0f, 1.0f, 1.0f }
126 #define HPDF_COLOR_BLACK            { 0.0f, 0.0f, 0.0f }
127
128 #else
130 #define HPDF_COLOR_DARK_RED          (HPDF_RGBColor) { 0.6f, 0.0f, 0.0f }
131 #define HPDF_COLOR_RED               (HPDF_RGBColor) { 1.0f, 0.0f, 0.0f }
132 #define HPDF_COLOR_LIGHT_GREEN      (HPDF_RGBColor) { 0.9f, 1.0f, 0.9f }
133 #define HPDF_COLOR_GREEN             (HPDF_RGBColor) { 0.4f, 0.9f, 0.4f }
134 #define HPDF_COLOR_DARK_GREEN       (HPDF_RGBColor) { 0.05f, 0.37f, 0.02f }
135 #define HPDF_COLOR_DARK_GRAY        (HPDF_RGBColor) { 0.2f, 0.2f, 0.2f }
136 #define HPDF_COLOR_LIGHT_GRAY       (HPDF_RGBColor) { 0.9f, 0.9f, 0.9f }
137 #define HPDF_COLOR_XLIGHT_GRAY      (HPDF_RGBColor) { 0.95f, 0.95f, 0.95f }
138 #define HPDF_COLOR_GRAY             (HPDF_RGBColor) { 0.5f, 0.5f, 0.5f }
139 #define HPDF_COLOR_SILVER            (HPDF_RGBColor) { 0.75f, 0.75f, 0.75f }
140 #define HPDF_COLOR_LIGHT_BLUE       (HPDF_RGBColor) { 1.0f, 1.0f, 0.9f }
141 #define HPDF_COLOR_BLUE             (HPDF_RGBColor) { 0.0f, 0.0f, 1.0f }
142 #define HPDF_COLOR_DARK_BLUE        (HPDF_RGBColor) { 0.0f, 0.0f, 0.6f }
143 #define HPDF_COLOR_WHITE            (HPDF_RGBColor) { 1.0f, 1.0f, 1.0f }
144 #define HPDF_COLOR_BLACK            (HPDF_RGBColor) { 0.0f, 0.0f, 0.0f }
145
146 #endif
147
148 #define HPDF_COLOR_ORANGE             HPDF_RGB_CONVERT(0xF5, 0xD0, 0x98);
149 #define HPDF_COLOR_ALMOST_BLACK      HPDF_RGB_CONVERT(0x14, 0x14, 0x14);
150
151 #define DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR 0.5
152
153
154 #define HPDFTBL_DEFAULT_TARGET_ENCODING "ISO8859-4"
155
156 #define HPDFTBL_DEFAULT_SOURCE_ENCODING "UTF-8"
157
158
159 #define A4PAGE_HEIGHT_CM 29.7
160
161 #define A4PAGE_WIDTH_CM 21.0
162
163 #define A3PAGE_HEIGHT_CM 42.0
164
165 #define A3PAGE_WIDTH_CM 29.7
166
167 #define LETTERRPAGE_HEIGHT_CM 27.9
168
169 #define LETTERRPAGE_WIDTH_CM 21.6
170
171 #define LEGALPAGE_HEIGHT_CM 35.6
172
173 #define LEGALPAGE_WIDTH_CM 21.6
174
175 #define HPDFTBL_END_CELLSPECS {0, 0, 0, 0, 0, 0, 0, 0, 0}
176
177 #define HPDF_COLOR_FROMRGB(r, g, b) (HPDF_RGBColor){(r)/255.0, (g)/255.0, (b)/255.0}
178
179 #define HPDFTBL_MIN_CALCULATED_PERCENT_CELL_WIDTH 2.0
180
181 #define hpdftbl_cm2dpi(c) (((HPDF_REAL)(c))/2.54*72)
182
183

```



```

243 #define _HPDFTBL_SET_ERR(t, err, r, c) do
    {hpdftbl_err_code=err;hpdftbl_err_row=r;hpdftbl_err_col=c;hpdftbl_err_lineno=__LINE__;hpdftbl_err_file=__FILE__;
    if (hpdftbl_err_handler){hpdftbl_err_handler(t,r,c,err);}} while(0)
244
251 #define _HPDFTBL_SET_ERR_EXTRA(info) do
    {strncpy(hpdftbl_err_extrainfo,info,1023);hpdftbl_err_extrainfo[1023]=0;} while(0)
252
256 #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)
257
261 #define _HPDFTBL_IDX(r, c) (r*t->cols+c)
262
270 typedef enum hpdftbl_text_align {
271     LEFT = 0,
272     CENTER = 1,
273     RIGHT = 2
274 } hpdftbl_text_align_t;
275
281 typedef struct text_style {
282     char *font;
283     HPDF_REAL fsize;
284     HPDF_RGBColor color;
285     HPDF_RGBColor background;
286     hpdftbl_text_align_t halign;
287 } hpdf_text_style_t;
288
297 typedef char *(*hpdftbl_content_callback_t)(void *, size_t, size_t);
298
308 typedef void (*hpdftbl_canvas_callback_t)(HPDF_Doc, HPDF_Page, void *, size_t, size_t, HPDF_REAL,
    HPDF_REAL, HPDF_REAL,
309                                     HPDF_REAL);
310
320 typedef _Bool (*hpdftbl_content_style_callback_t)(void *, size_t, size_t, char *content,
    hpdf_text_style_t *);
321
330 typedef enum hpdftbl_dashstyle {
331     LINE_SOLID ,
332     LINE_DOT1 ,
333     LINE_DOT2 ,
334     LINE_DOT3 ,
335     LINE_DOT4 ,
336     LINE_DASH1 ,
337     LINE_DASH2 ,
338     LINE_DASH3 ,
339     LINE_DASH4 ,
340     LINE_DASH5 ,
341     LINE_DASHDOT1 ,
342     LINE_DASHDOT2
343 } hpdftbl_line_dashstyle_t;
344
350 typedef struct grid_style {
351     HPDF_REAL width;
352     HPDF_RGBColor color;
353     hpdftbl_line_dashstyle_t line_dashstyle;
354 } hpdftbl_grid_style_t;
355
363 struct hpdftbl_cell {
364     char *label;
365     char *content;
366     size_t colspan;
367     size_t rowspan;
368     HPDF_REAL height;
369     HPDF_REAL width;
370     HPDF_REAL delta_x;
371     HPDF_REAL delta_y;
372     HPDF_REAL textwidth;
373     hpdftbl_content_callback_t content_cb;
374     hpdftbl_content_callback_t label_cb;
375     hpdftbl_content_style_callback_t style_cb;
376     hpdftbl_canvas_callback_t canvas_cb;
377     hpdf_text_style_t content_style;
378     struct hpdftbl_cell *parent_cell;
379 };
380
403 typedef struct hpdftbl_cell hpdftbl_cell_t;
404
413 struct hpdftbl {
414     HPDF_Doc pdf_doc;
415     HPDF_Page pdf_page;
416     size_t cols;
417     size_t rows;
418     HPDF_REAL posx;
419     HPDF_REAL posy;
420     HPDF_REAL height;
421     HPDF_REAL minheight;
422     HPDF_REAL bottom_vmargin_factor;
423     HPDF_REAL width;

```

```

435     void *tag;
437     char *title_txt;
439     hpdf_text_style_t title_style;
441     hpdf_text_style_t header_style;
443     _Bool use_header_row;
445     hpdf_text_style_t label_style;
447     _Bool use_cell_labels;
449     _Bool use_label_grid_style;
451     hpdftbl_content_callback_t label_cb;
453     hpdf_text_style_t content_style;
455     hpdftbl_content_callback_t content_cb;
457     hpdftbl_content_style_callback_t content_style_cb;
459     hpdftbl_canvas_callback_t canvas_cb;
461     hpdftbl_cell_t *cells;
463     hpdftbl_grid_style_t outer_grid;
465     hpdftbl_grid_style_t inner_vgrid;
467     hpdftbl_grid_style_t inner_hgrid;
469     hpdftbl_grid_style_t inner_tgrid;
473     _Bool use_zebra;
477     int zebra_phase;
479     HPDF_RGBColor zebra_color1;
481     HPDF_RGBColor zebra_color2;
483     float *col_width_percent;
484 };
485
494 typedef struct hpdftbl *hpdftbl_t;
495
505 typedef void (*hpdftbl_callback_t) (hpdftbl_t);
506
516 typedef struct hpdftbl_cell_spec {
518     size_t row;
520     size_t col;
522     unsigned rowspan;
524     unsigned colspan;
526     char *label;
528     hpdftbl_content_callback_t content_cb;
530     hpdftbl_content_callback_t label_cb;
532     hpdftbl_content_style_callback_t style_cb;
534     hpdftbl_canvas_callback_t canvas_cb;
535 } hpdftbl_cell_spec_t;
536
543 typedef struct hpdftbl_spec {
545     char *title;
547     _Bool use_header;
549     _Bool use_labels;
551     _Bool use_labelgrid;
553     size_t rows;
555     size_t cols;
557     HPDF_REAL xpos;
559     HPDF_REAL ypos;
561     HPDF_REAL width;
563     HPDF_REAL height;
565     hpdftbl_content_callback_t content_cb;
567     hpdftbl_content_callback_t label_cb;
569     hpdftbl_content_style_callback_t style_cb;
574     hpdftbl_callback_t post_cb;
576     hpdftbl_cell_spec_t *cell_spec;
577 } hpdftbl_spec_t;
578
585 typedef struct hpdftbl_theme {
587     hpdf_text_style_t content_style;
589     hpdf_text_style_t label_style;
591     hpdf_text_style_t header_style;
593     hpdf_text_style_t title_style;
595     hpdftbl_grid_style_t outer_border;
597     _Bool use_labels;
599     _Bool use_label_grid_style;
601     _Bool use_header_row;
603     hpdftbl_grid_style_t inner_vborder;
605     hpdftbl_grid_style_t inner_hborder;
607     hpdftbl_grid_style_t inner_tborder;
609     _Bool use_zebra;
611     int zebra_phase;
613     HPDF_RGBColor zebra_color1;
615     HPDF_RGBColor zebra_color2;
617     HPDF_REAL bottom_vmargin_factor;
618 } hpdftbl_theme_t;
619
627 typedef void (*hpdftbl_error_handler_t) (hpdftbl_t, int, int, int);
628
629 extern hpdftbl_error_handler_t hpdftbl_err_handler ;
630
631 /*
632  * Table creation and destruction function
633  */
634 hpdftbl_t
635 hpdftbl_create(size_t rows, size_t cols);

```

```

636
637 hpdftbl_t
638 hpdftbl_create_title(size_t rows, size_t cols, char *title);
639
640 int
641 hpdftbl_stroke(HPDF_Doc pdf,
642                HPDF_Page page, hpdftbl_t t,
643                HPDF_REAL xpos, HPDF_REAL ypos,
644                HPDF_REAL width, HPDF_REAL height);
645
646 int
647 hpdftbl_stroke_from_data(HPDF_Doc pdf_doc, HPDF_Page pdf_page, hpdftbl_spec_t *tbl_spec, hpdftbl_theme_t
        *theme);
648
649 int
650 hpdftbl_destroy(hpdftbl_t t);
651
652 int
653 hpdftbl_get_last_auto_height(HPDF_REAL *height);
654
655 void
656 hpdftbl_set_anchor_top_left(_Bool anchor);
657
658 _Bool
659 hpdftbl_get_anchor_top_left(void);
660
661 /*
662  * Table error handling functions
663  */
664 hpdftbl_error_handler_t
665 hpdftbl_set_errhandler(hpdftbl_error_handler_t);
666
667 const char *
668 hpdftbl_get_errstr(int err);
669
670 const char *
671 hpdftbl_hpdpf_get_errstr(HPDF_STATUS err_code);
672
673 int
674 hpdftbl_get_last_errcode(const char **errstr, int *row, int *col);
675
676 void
677 hpdftbl_get_last_err_file(int *lineno, char **file, char **extrainfo);
678
679 void
680 hpdftbl_default_table_error_handler(hpdftbl_t t, int r, int c, int err);
681
682 /*
683  * Theme handling functions
684  */
685 int
686 hpdftbl_apply_theme(hpdftbl_t t, hpdftbl_theme_t *theme);
687
688 hpdftbl_theme_t *
689 hpdftbl_get_default_theme(void);
690
691 int
692 hpdftbl_destroy_theme(hpdftbl_theme_t *theme);
693
694 /*
695  * Table layout adjusting functions
696  */
697
698 void
699 hpdftbl_set_bottom_vmargin_factor(hpdftbl_t t, HPDF_REAL f);
700
701 int
702 hpdftbl_set_min_rowheight(hpdftbl_t t, float h);
703
704 int
705 hpdftbl_set_colwidth_percent(hpdftbl_t t, size_t c, float w);
706
707 int
708 hpdftbl_clear_spanning(hpdftbl_t t);
709
710 int
711 hpdftbl_set_cellspan(hpdftbl_t t, size_t r, size_t c, size_t rowspan, size_t colspan);
712
713 /*
714  * Table style handling functions
715  */
716 int
717 hpdftbl_set_zebra(hpdftbl_t t, _Bool use, int phase);
718
719 int
720 hpdftbl_set_zebra_color(hpdftbl_t t, HPDF_RGBColor z1, HPDF_RGBColor z2);
721

```

```

722 int
723 hpdftbl_use_labels(hpdftbl_t t, _Bool use);
724
725 int
726 hpdftbl_use_labelgrid(hpdftbl_t t, _Bool use);
727
728 int
729 hpdftbl_set_background(hpdftbl_t t, HPDF_RGBColor background);
730
731 int
732 hpdftbl_set_inner_tgrid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color,
733     hpdftbl_line_dashstyle_t dashstyle);
734
735 int
736 hpdftbl_set_inner_vgrid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color,
737     hpdftbl_line_dashstyle_t dashstyle);
738
739 int
740 hpdftbl_set_inner_hgrid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color,
741     hpdftbl_line_dashstyle_t dashstyle);
742
743 int
744 hpdftbl_set_outer_grid_style(hpdftbl_t t, HPDF_REAL width, HPDF_RGBColor color, hpdftbl_line_dashstyle_t
745     dashstyle);
746
747 int
748 hpdftbl_set_header_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
749     background);
750
751 int
752 hpdftbl_set_header_halign(hpdftbl_t t, hpdftbl_text_align_t align);
753
754 int
755 hpdftbl_use_header(hpdftbl_t t, _Bool use);
756
757 int
758 hpdftbl_set_label_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
759     background);
760
761 int
762 hpdftbl_set_row_content_style(hpdftbl_t t, size_t r, char *font, HPDF_REAL fsize, HPDF_RGBColor color,
763     HPDF_RGBColor background);
764
765 int
766 hpdftbl_set_col_content_style(hpdftbl_t t, size_t c, char *font, HPDF_REAL fsize, HPDF_RGBColor color,
767     HPDF_RGBColor background);
768
769 int
770 hpdftbl_set_content_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
771     background);
772
773 int
774 hpdftbl_set_cell_content_style(hpdftbl_t t, size_t r, size_t c, char *font, HPDF_REAL fsize,
775     HPDF_RGBColor color,
776     HPDF_RGBColor background);
777
778 int
779 hpdftbl_set_title_style(hpdftbl_t t, char *font, HPDF_REAL fsize, HPDF_RGBColor color, HPDF_RGBColor
780     background);
781
782 /*
783  * Table content handling
784  */
785
786 int
787 hpdftbl_set_cell(hpdftbl_t t, int r, int c, char *label, char *content);
788
789 int
790 hpdftbl_set_tag(hpdftbl_t t, void *tag);
791
792 int
793 hpdftbl_set_title(hpdftbl_t t, char *title);
794
795 int
796 hpdftbl_set_title_halign(hpdftbl_t t, hpdftbl_text_align_t align);
797
798 int
799 hpdftbl_set_labels(hpdftbl_t t, char **labels);
800
801 int
802 hpdftbl_set_content(hpdftbl_t t, char **content);
803
804 /*
805  * Table callback functions

```

```

799  */
800  int
801  hpdftbl_set_content_cb(hpdftbl_t t, hpdftbl_content_callback_t cb);
802
803  int
804  hpdftbl_set_cell_content_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb);
805
806  int
807  hpdftbl_set_label_cb(hpdftbl_t t, hpdftbl_content_callback_t cb);
808
809  int
810  hpdftbl_set_cell_label_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_callback_t cb);
811
812  int
813  hpdftbl_set_canvas_cb(hpdftbl_t t, hpdftbl_canvas_callback_t cb);
814
815  int
816  hpdftbl_set_cell_canvas_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_canvas_callback_t cb);
817
818  int
819  hpdftbl_set_content_style_cb(hpdftbl_t t, hpdftbl_content_style_callback_t cb);
820
821  int
822  hpdftbl_set_cell_content_style_cb(hpdftbl_t t, size_t r, size_t c, hpdftbl_content_style_callback_t cb);
823
824  /*
825   * Table dynamic callback functions
826   */
827  void
828  hpdftbl_set_dlhandle(void *);
829
830  int
831  hpdftbl_set_content_dyncb(hpdftbl_t, char *);
832
833  int
834  hpdftbl_set_cell_content_dyncb(hpdftbl_t, size_t, size_t, char *);
835
836  int
837  hpdftbl_set_label_dyncb(hpdftbl_t, char *);
838
839  int
840  hpdftbl_set_cell_label_dyncb(hpdftbl_t, size_t, size_t, char *);
841
842  int
843  hpdftbl_set_content_style_dyncb(hpdftbl_t, char *);
844
845  int
846  hpdftbl_set_cell_content_style_dyncb(hpdftbl_t, size_t, size_t, char *);
847
848  int
849  hpdftbl_set_cell_canvas_dyncb(hpdftbl_t, size_t, size_t, char *);
850
851  /*
852   * Text encoding
853   */
854  void
855  hpdftbl_set_text_encoding(char *target, char *source);
856
857  int
858  hpdftbl_encoding_text_out(HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, char *text);
859
860  /*
861   * Misc utility and widget functions
862   */
863
864  void
865  HPDF_RoundedCornerRectangle(HPDF_Page page, HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL
      height,
866                             HPDF_REAL rad);
867
868  void
869  hpdftbl_stroke_grid(HPDF_Doc pdf, HPDF_Page page);
870
871  void
872  hpdftbl_table_widget_letter_buttons(HPDF_Doc doc, HPDF_Page page,
873                                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
874                                     HPDF_RGBColor on_color, HPDF_RGBColor off_color,
875                                     HPDF_RGBColor on_background, HPDF_RGBColor off_background,
876                                     HPDF_REAL fsize,
877                                     const char *letters, _Bool *state);
878
879  void
880  hpdftbl_widget_slide_button(HPDF_Doc doc, HPDF_Page page,
881                              HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height, _Bool
      state);
882
883  void

```

```

884 hpdfdbl_widget_hbar(HPDF_Doc doc, HPDF_Page page,
885                     HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
886                     HPDF_RGBColor color, float val, _Bool hide_val);
887
888 void
889 hpdfdbl_widget_segment_hbar(HPDF_Doc doc, HPDF_Page page,
890                             HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
891                             size_t num_segments, HPDF_RGBColor on_color, double val_percent,
892                             _Bool hide_val);
893
894 void
895 hpdfdbl_widget_strength_meter(HPDF_Doc doc, HPDF_Page page,
896                               HPDF_REAL xpos, HPDF_REAL ypos, HPDF_REAL width, HPDF_REAL height,
897                               size_t num_segments, HPDF_RGBColor on_color, size_t num_on_segments);
898
899 int
900 hpdfdbl_stroke_pdfdoc(HPDF_Doc pdf_doc, char *file);
901
902 /*
903  * Internal functions
904  */
905 _Bool
906 chkdbl(hpdfdbl_t, size_t, size_t);
907
908 #ifdef __cplusplus
909 }
910 #endif
911
912 #endif /* hpdfdbl_H */

```

15.11 hpdfdbl_callback.c File Reference

Routines for callback function 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 <libgen.h>
#include <sys/stat.h>
#include <dlfcn.h>
#include "hpdfdbl.h"

```

Functions

- void [hpdfdbl_set_dlhandle](#) (void *handle)
Set the handle for scope of dynamic function search.
- int [hpdfdbl_set_content_cb](#) ([hpdfdbl_t](#) t, [hpdfdbl_content_callback_t](#) cb)
Set table content callback.
- int [hpdfdbl_set_cell_content_cb](#) ([hpdfdbl_t](#) t, size_t r, size_t c, [hpdfdbl_content_callback_t](#) cb)
Set cell content callback.
- int [hpdfdbl_set_cell_label_cb](#) ([hpdfdbl_t](#) t, size_t r, size_t c, [hpdfdbl_content_callback_t](#) cb)
Set cell label callback.
- int [hpdfdbl_set_cell_canvas_cb](#) ([hpdfdbl_t](#) t, size_t r, size_t c, [hpdfdbl_canvas_callback_t](#) cb)
Set cell canvas callback.
- int [hpdfdbl_set_label_cb](#) ([hpdfdbl_t](#) t, [hpdfdbl_content_callback_t](#) cb)
Set table label callback.
- int [hpdfdbl_set_canvas_cb](#) ([hpdfdbl_t](#) t, [hpdfdbl_canvas_callback_t](#) cb)
Set cell canvas callback.

- `int hpdftbl_set_content_dyncb (hpdftbl_t t, char *cb_name)`
Specify dynamic (late) loading callback content function.
- `int hpdftbl_set_label_dyncb (hpdftbl_t t, char *cb_name)`
Specify dynamic (late) loading callback for table label function.
- `int hpdftbl_set_cell_label_dyncb (hpdftbl_t t, size_t r, size_t c, char *cb_name)`
Specify dynamic (late) loading callback for cell label function.
- `int hpdftbl_set_content_style_dyncb (hpdftbl_t t, char *cb_name)`
Specify dynamic (late) loading callback for table style function.
- `int hpdftbl_set_cell_content_style_dyncb (hpdftbl_t t, size_t r, size_t c, char *cb_name)`
Specify dynamic (late) loading callback for cell style function.
- `int hpdftbl_set_cell_content_dyncb (hpdftbl_t t, size_t r, size_t c, char *cb_name)`
Specify dynamic (late) loading callback cell content function.
- `int hpdftbl_set_cell_canvas_dyncb (hpdftbl_t t, size_t r, size_t c, char *cb_name)`
Specify dynamic (late) loading callback cell canvas function.
- `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.

15.11.1 Detailed Description

Routines for callback function for flexible table drawing with HPDF library.

Author

Johan Persson (johan162@gmail.com)

Copyright (C) 2022 Johan Persson

See also

LICENSE

15.11.2 Function Documentation

15.11.2.1 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

[hpdfdbl_set_cell_canvas_cb\(\)](#)

15.11.2.2 hpdfdbl_set_cell_canvas_cb()

```
int hpdfdbl_set_cell_canvas_cb (
    hpdfdbl_t t,
    size_t r,
    size_t c,
    hpdfdbl_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\(\)](#).

15.11.2.3 hpdftbl_set_cell_canvas_dyncb()

```
int hpdftbl_set_cell_canvas_dyncb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback cell canvas function.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as canvas callback. This function must follow the signature of a callback function as specified in <code>hpdftbl_canvas_callback_t</code>

Returns

See also

[hpdftbl_set_cell_canvas_cb\(\)](#), [hpdftbl_canvas_callback_t](#)

15.11.2.4 hpdftbl_set_cell_content_cb()

```
int hpdftbl_set_cell_content_cb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    hpdftbl_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\(\)](#).

15.11.2.5 hpdfdbl_set_cell_content_dyncb()

```
int hpdfdbl_set_cell_content_dyncb (
    hpdfdbl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback cell content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdfdbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in <code>hpdfdbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdfdbl_set_content_cb\(\)](#), [hpdfdbl_content_callback_t](#)

15.11.2.6 hpdfdbl_set_cell_content_style_cb()

```
int hpdfdbl_set_cell_content_style_cb (
    hpdfdbl_t t,
```

```

size_t r,
size_t c,
hpdfctl_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

[hpdfctl_set_content_style_cb\(\)](#)

Referenced by [hpdfctl_stroke_from_data\(\)](#).

15.11.2.7 hpdfctl_set_cell_content_style_dyncb()

```

int hpdfctl_set_cell_content_style_dyncb (
    hpdfctl_t t,
    size_t r,
    size_t c,
    char * cb_name )

```

Specify dynamic (late) loading callback for cell style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdfctl_content_style_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdfctl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdfctl_content_style_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_cell_content_style_cb\(\)](#), [hpdftbl_content_style_callback_t](#)

15.11.2.8 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\(\)](#).

15.11.2.9 hpdftbl_set_cell_label_dyncb()

```
int hpdftbl_set_cell_label_dyncb (
    hpdftbl_t t,
    size_t r,
    size_t c,
    char * cb_name )
```

Specify dynamic (late) loading callback for cell label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdfctl_content_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdfctl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>r</i>	Cell row
<i>c</i>	Cell column
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdfdbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdfdbl_set_cell_label_cb\(\)](#), [hpdfdbl_content_callback_t](#)

15.11.2.10 hpdfdbl_set_content_cb()

```
int hpdfdbl_set_content_cb (  
    hpdfdbl_t t,  
    hpdfdbl_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

[hpdfdbl_set_cell_content_cb\(\)](#)

Referenced by [hpdfdbl_stroke_from_data\(\)](#).

15.11.2.11 `hpdtbl_set_content_dyncb()`

```
int hpdtbl_set_content_dyncb (
    hpdtbl_t t,
    char * cb_name )
```

Specify dynamic (late) loading callback content function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol.

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the `hpdtbl_get_last_err_file()` to read it back.

Parameters

<i>t</i>	Table handle
<i>cb_name</i>	Name of the function to be used as content callback. This function must follow the signature of a callback function as specified in <code>hpdtbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdtbl_set_content_cb\(\)](#), [hpdtbl_content_callback_t](#)

15.11.2.12 `hpdtbl_set_content_style_cb()`

```
int hpdtbl_set_content_style_cb (
    hpdtbl_t t,
    hpdtbl_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\(\)](#).

15.11.2.13 hpdftbl_set_content_style_dyncb()

```
int hpdftbl_set_content_style_dyncb (
    hpdftbl_t t,
    char * cb_name )
```

Specify dynamic (late) loading callback for table style function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the `dlsym()` loading of external symbols. For the external symbol to be found it can not be defined as a `static` symbol. The callback function must have the signature defined by `hpdftbl_content_style_callback_t`

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdftbl_content_style_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdftbl_set_content_style_cb\(\)](#), [hpdftbl_content_style_callback_t](#)

15.11.2.14 hpdftbl_set_dlhandle()

```
void hpdftbl_set_dlhandle (
    void * handle )
```

Set the handle for scope of dynamic function search.

When using late binding (some `os_dyncb()` functions) the scope for where the runtime searches for the functions can be specified as is discussed in `man 3 dlsym`. By default the library uses `dl_handle` which make the library first searches the current image and then all images it was built against.

If the dynamic callbacks are located in a runtime loaded library then the handle returned by `dlopen()` must be specified as the function will not be found otherwise.

Parameters

<i>handle</i>	Predefined values or the handle returned by dlopen() (see man dlopen)
---------------	---

15.11.2.15 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\(\)](#).

15.11.2.16 hpdftbl_set_label_dyncb()

```
int hpdftbl_set_label_dyncb (
    hpdftbl_t t,
    char * cb_name )
```

Specify dynamic (late) loading callback for table label function.

The dynamic loading of callback function is a runtime binding of the named function as a callback. The library uses the dlsym() loading of external symbols. For the external symbol to be found it can not be defined as a static symbol. The callback function must have the signature defined by hpdftbl_content_callback_t

In case of error the `extrainfo` extra information is set to the name of the callback which failed to be resolved at run time. This can be retrieved in an error handler by using the [hpdftbl_get_last_err_file\(\)](#) to read it back.

Parameters

<i>t</i>	Table handle
<i>cb_name</i>	Name of the function to be used as label callback. This function must follow the signature of a callback function as specified in <code>hpdf_tbl_content_callback_t</code> .

Returns

-1 on failure, 0 on success

See also

[hpdf_tbl_set_label_cb\(\)](#), [hpdf_tbl_content_callback_t](#)

15.12 hpdf_tbl_errstr.c File Reference

Utility module to translate HPDF error codes to human readable strings.

```
#include <hpdf.h>
```

Data Structures

- struct [hpdf_tbl_errcode_entry](#)
An entry in the error string table.

Functions

- 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.

15.12.1 Detailed Description

Utility module to translate HPDF error codes to human readable strings.

15.12.2 Function Documentation

15.12.2.1 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

<i>err_code</i>	The error code
-----------------	----------------

Returns

A pointer to an error string, NULL if the error code is invalid

See also

[hpdfdbl_get_errstr\(\)](#)

15.13 hpdfdbl_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 [hpdfdbl_stroke_grid](#) (HPDF_Doc pdf, HPDF_Page page)

15.13.1 Detailed Description

Create a grid on a document for positioning.

15.13.2 Function Documentation

15.13.2.1 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

15.14 hpdftbl_theme.c File Reference

Functions for theme handling.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <hpdf.h>
#include "hpdftbl.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, 0.97f}, [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** ([hpdftbl_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** ([hpdftbl_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** ([hpdftbl_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.

15.14.1 Detailed Description

Functions for theme handling.

Author

Johan Persson (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.

15.14.2 Macro Definition Documentation

15.14.2.1 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\(\)](#)

15.14.2.6 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\(\)](#)

15.14.2.7 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

15.14.2.8 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

15.14.3 Function Documentation

15.14.3.1 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

[hpdfdbl_get_default_theme\(\)](#)

Referenced by [hpdfdbl_create_title\(\)](#), and [hpdfdbl_stroke_from_data\(\)](#).

15.14.3.2 hpdfdbl_destroy_theme()

```
int hpdfdbl_destroy_theme (
    hpdfdbl_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 [hpdfdbl_create_title\(\)](#).

15.14.3.3 hpdfdbl_get_default_theme()

```
hpdfdbl_theme_t * hpdfdbl_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 [hpdfdbl_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

[hpdfctl_apply_theme\(\)](#)

Referenced by [hpdfctl_create_title\(\)](#).

15.15 hpdfctl_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 "hpdfctl.h"
```

Macros

- `#define TRUE 1`
- `#define FALSE 0`

Functions

- void [hpdfctl_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 [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](#) (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 [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.
- 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.

15.15.1 Detailed Description

Support for drawing widgets.

15.15.2 Macro Definition Documentation

15.15.2.1 FALSE

```
#define FALSE 0
```

C Boolean false value

15.15.2.2 TRUE

```
#define TRUE 1
```

C Boolean truth value

15.15.3 Function Documentation

15.15.3.1 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)

15.15.3.2 hpdftbl_widget_hbar()

```
void hpdftbl_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

15.15.3.3 `hpdf_tbl_widget_segment_hbar()`

```
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 )
```

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

15.15.3.4 `hpdf_tbl_widget_slide_button()`

```
void hpdf_tbl_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

15.15.3.5 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

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 simulated exception handling
jmp_buf _hpdftbl_jmp_env;
#include "unit_test.inc.h"
// 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++) {
            if (defined _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), "Content %zu", cnt);
                content[cnt] = strdup(buff);
            #endif
            cnt++;
        }
    }
}
```

```

#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
#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 (run_as_unit_test || -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 ( ! run_as_unit_test ) {
        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,label-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
// Add another page in the document
static void
add_a4page(void) {
    pdf_page = HPDF_AddPage(pdf_doc);
}

```

```

    HPDF_Page_SetSize(pdf_page, HPDF_PAGE_SIZE_A4, HPDF_PAGE_PORTRAIT);
}
void
ex_ttbl1(void) {
    int num_rows = 5;
    int num_cols = 4;
    char *table_title = "Example 1: Basic table with default theme";
    hpdf_ttbl_t t = hpdf_ttbl_create_title(num_rows, num_cols, table_title);
    hpdf_ttbl_set_content(t, content);
    hpdf_ttbl_set_labels(t, labels);
    hpdf_ttbl_use_labels(t, FALSE);
    //hpdf_ttbl_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 = hpdf_ttbl_cm2dpi(2);
    HPDF_REAL ypos = hpdf_ttbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdf_ttbl_cm2dpi(15);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdf_ttbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_ttbl2(void) {
    int num_rows = 5;
    int num_cols = 4;
    char *table_title = "Example 2: Basic table with adjusted font styles";
    hpdf_ttbl_t t = hpdf_ttbl_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;
    hpdf_ttbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                             title_bg_color);
    hpdf_ttbl_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;
    hpdf_ttbl_set_content_style(t, HPDF_FF_COURIER_BOLD, 10,
                                content_text_color, content_bg_color);
    hpdf_ttbl_set_content(t, content);
    hpdf_ttbl_set_labels(t, labels);
    hpdf_ttbl_use_labels(t, TRUE);
    hpdf_ttbl_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 = hpdf_ttbl_cm2dpi(2);
    HPDF_REAL ypos = hpdf_ttbl_cm2dpi(A4PAGE_HEIGHT_CM - 4);
    HPDF_REAL width = hpdf_ttbl_cm2dpi(15);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdf_ttbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}
void
ex_ttbl3(void) {
    int num_rows = 9;
    int num_cols = 4;
    char *table_title =
        "Example 3: Table cell spannings and full grid and header";
    hpdf_ttbl_t t = hpdf_ttbl_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;
    hpdf_ttbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                             title_bg_color);
    hpdf_ttbl_set_title_halign(t, CENTER);
    // Use specially formatted header row
    hpdf_ttbl_use_header(t, TRUE);
    // Use full grid and not just the short labelgrid
    hpdf_ttbl_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;
    hpdf_ttbl_set_content_style(t, HPDF_FF_COURIER_BOLD, 10,
                                content_text_color, content_bg_color);
    hpdf_ttbl_set_content(t, content);
    hpdf_ttbl_set_labels(t, labels);
    hpdf_ttbl_use_labels(t, TRUE);
    // Spanning for the header row (row==0)
    // Span cell=(0,1) one row and three columns
    hpdf_ttbl_set_cellspan(t, 0, 1, 1, 3);
    // Span cell=(1,1) one row and three columns
    hpdf_ttbl_set_cellspan(t, 1, 1, 1, 3);
    // Span cell=(2,2) one row and two columns
    hpdf_ttbl_set_cellspan(t, 2, 2, 1, 2);
    // Span cell=(4,1) two rows and three columns
    hpdf_ttbl_set_cellspan(t, 4, 1, 2, 3);
}

```

```

// Span cell=(7,2) two rows and two columns
hpdf_tbl_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 = 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
hpdf_tbl_stroke(pdf_doc, pdf_page, t, xpos, ypos, width, height);
}

void
ex_tbl4(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";
    hpdf_tbl_t t = hpdf_tbl_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;
    hpdf_tbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                             title_bg_color);
    hpdf_tbl_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;
    hpdf_tbl_set_cell_content_style(t, 0, 0, HPDF_FF_COURIER_BOLD, 10,
                                     content_text_color, content_bg_color);
    hpdf_tbl_set_cell_content_style(t, 4, 3, HPDF_FF_COURIER_BOLD, 10,
                                     content_text_color, content_bg_color);
    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);
    // Span cell=(1,0) one row and two columns
    hpdf_tbl_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 = 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);
    }
}

void
ex_tbl5(void) {
    const int num_rows = 6;
    const int num_cols = 4;
    char *table_title = "Example 5: Using widgets in cells";
    hpdf_tbl_t t = hpdf_tbl_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;
    hpdf_tbl_set_title_style(t, HPDF_FF_HELVETICA_BOLD, 14, title_text_color,
                             title_bg_color);
    hpdf_tbl_set_title_halign(t, CENTER);
    hpdf_tbl_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:";
hpdftbl_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:";
hpdftbl_set_cell_canvas_cb(t, wrow, wcol, cb_draw_boxed_letter);
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);
// 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);
}
}
// Type for the pointer to example stroking functions "void fnc(void)"
typedef void (*t_func_tbl_stroke)(void);
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(_hpdftbl_jmp_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.
    run_as_unit_test = 2 == argc ;
    // Get some dummy data to fill the tables
    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();
    }
    #if !(defined __WIN32 || defined __WIN32__)
        example_page_header();
    #endif
    (*examples[i])();
}
if (-1 == stroke_to_file(pdf_doc, argc, argv) )
    return EXIT_FAILURE;
else
    return EXIT_SUCCESS;
}

```

16.2 tut_ex01.c

The very most basic table with API call to set content in each cell.

```

#include "unit_test.inc.h"
void
create_table_ex01(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    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
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex01, FALSE)

```

16.3 tut_ex02.c

Basic table with content data specified as an array.

```

#include "unit_test.inc.h"
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_content(&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
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex02, FALSE)

```

16.4 tut_ex02_1.c

Basic table with content data specified as an array.

```

#include "unit_test.inc.h"
void setup_dummy_content_with_header(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;
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    hpdf_tbl_use_header(tbl, TRUE);
    content_t content;
    setup_dummy_content_with_header(&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);
}
TUTEX_MAIN(create_table_ex02_1, FALSE)

```

16.5 tut_ex03.c

First example with API call to set content in each cell with added labels and shortened grid.

```

#include "unit_test.inc.h"
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";
hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
hpdf_tbl_set_cell(tbl, 0, 0, "Label 1", "Cell 0x0");
hpdf_tbl_set_cell(tbl, 0, 1, "Label 2", "Cell 0x1");
hpdf_tbl_set_cell(tbl, 1, 0, "Label 3", "Cell 1x0");
hpdf_tbl_set_cell(tbl, 1, 1, "Label 4", "Cell 1x1");
hpdf_tbl_use_labels(tbl, TRUE);
hpdf_tbl_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 = 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);
}
TUTEX_MAIN(create_table_ex03, FALSE)

```

16.6 tut_ex04.c

Specifying labels as data array.

```

#include "unit_test.inc.h"
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";
    hpdf_tbl_t tbl = hpdf_tbl_create(num_rows, num_cols);
    content_t content, labels;
    setup_dummy_content_label(&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);
}
TUTEX_MAIN(create_table_ex04, FALSE)

```

16.7 tut_ex05.c

Set content data specified as an array with added labels and shortened grid.

```

#include "unit_test.inc.h"
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_content_label(&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);
}
TUTEX_MAIN(create_table_ex05, FALSE)

```

16.8 tut_ex06.c

Use content to set content and labels.

```
#include "unit_test.inc.h"
#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
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        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";
    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_content_cb(tbl, cb_content);
    hpdf_tbl_set_label_cb(tbl, cb_labels);
    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
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex06, FALSE)
```

16.9 tut_ex07.c

Expand cells over multiple columns and rows.

```
#include "unit_test.inc.h"
#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
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        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);
}
TUTEX_MAIN(create_table_ex07, FALSE)

```

16.10 tut_ex08.c

Adjust column width and expand cells over multiple columns and rows.

```

#include "unit_test.inc.h"
#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
static char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];

```



```

    if ( ! run_as_unit_test ) {
        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";
    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_content_cb(tbl, cb_content);
    hpdf_tbl_set_label_cb(tbl, cb_labels);
    hpdf_tbl_set_cell_content_cb(tbl, 0, 0, cb_date);
    hpdf_tbl_set_cellspan(tbl, 0, 0, 1, 3);
    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(17);
    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);
}

TUTEX_MAIN(create_table_ex08, FALSE)

```

16.11 tut_ex09.c

Adjusting font style with a callback.

```

#include "unit_test.inc.h"

#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

_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;
}

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;
    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
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

TUTEX_MAIN(create_table_ex09, FALSE)

```

16.12 tut_ex10.c

Adjust column widths and add error handler.

```

#include "unit_test.inc.h"

void
create_table_ex10(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_set_colwidth_percent(tbl, 0, 30);
    hpdf_tbl_set_colwidth_percent(tbl, 1, 30);
    content_t content;
    setup_dummy_content(&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 - 4);
    HPDF_REAL height = 0; // Calculate height automatically
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}

TUTEX_MAIN(create_table_ex10, FALSE)

```

16.13 tut_ex11.c

Table with header row and error handler.

```
#include "unit_test.inc.h"
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_content(&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);
}
TUTEX_MAIN(create_table_ex11, FALSE)
```

16.14 tut_ex12.c

Table with header row and error handler.

```
#include "unit_test.inc.h"
void
create_table_ex12(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_content(&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);
}
TUTEX_MAIN(create_table_ex12, FALSE)
```

16.15 tut_ex13_1.c

Defining a table with a data structure for the table.

```
#include "unit_test.inc.h"
#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
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;
}
#endif
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
hpdftbl_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=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
    .width=hpdftbl_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) {
    hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
}
TUTEX_MAIN(create_table_ex13_1, FALSE)

```

16.16 tut_ex13_2.c

Defining a table with a data structure for table and cells.

```

#include "unit_test.inc.h"
#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
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},
        HPDFTBL_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=hpdftbl_cm2dpi(1), .ypos=hpdftbl_cm2dpi(A4PAGE_HEIGHT_CM-2),
    // width and height
    .width=hpdftbl_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) {
    hpdftbl_stroke_from_data(pdf_doc, pdf_page, &tbl_spec, NULL);
}
TUTEX_MAIN(create_table_ex13_2, FALSE)

```

16.17 tut_ex14.c

Defining a table with widgets.

```

#include "unit_test.inc.h"
#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
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 ( ! run_as_unit_test ) {
        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;
    hpdftbl_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";
    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);
}

TUTEX_MAIN(create_table_ex14, FALSE)

```

16.18 tut_ex15.c

Defining a table with zebra lines.

```

#include "unit_test.inc.h"
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_content(&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);
}
TUTEX_MAIN(create_table_ex15, FALSE)

```

16.19 tut_ex15_1.c

Defining a table with zebra lines and different phase.

```

#include "unit_test.inc.h"
void
create_table_ex15_1(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_content(&content, num_rows, num_cols);
    hpdftbl_set_content(tbl, content);
    hpdftbl_set_zebra(tbl, TRUE, 1);
    hpdftbl_set_inner_hgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY, LINE_SOLID);
    hpdftbl_set_inner_tgrid_style(tbl, 0.5, HPDF_COLOR_XLIGHT_GRAY, LINE_DOT1);
    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);
}
TUTEX_MAIN(create_table_ex15_1, FALSE)

```

16.20 tut_ex20.c

Defining a table and adjusting the gridlines.

```

#include "unit_test.inc.h"
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_content_label(&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);
}
TUTEX_MAIN(create_table_ex20, FALSE)

```

16.21 tut_ex30.c

Defining a table using dynamic callbacks

```
#include "dlfcn.h"
#include "unit_test.inc.h"
#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"
#pragma GCC diagnostic ignored "-Wunused-function"
#endif
char *
cb_date(void *tag, size_t r, size_t c) {
    static char buf[64];
    if ( ! run_as_unit_test ) {
        time_t t = time(NULL);
        ctime_r(&t, buf);
        return buf;
    } else {
        return "Wed May 4 19:01:01 2022";
    }
}
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;
}
char *
cb_labels(void *tag, size_t r, size_t c) {
    static char buf[32];
    if (0==r && 0==c) {
        snprintf(buf, sizeof buf, "Date created:");
    } else {
        snprintf(buf, sizeof buf, "Label %zux%zu:", r, c);
    }
    return buf;
}
#ifdef _MSC_VER
#pragma GCC diagnostic pop
#endif
void
create_table_ex30(HPDF_Doc pdf_doc, HPDF_Page pdf_page) {
    const size_t num_rows = 2;
    const size_t num_cols = 2;
    char *table_title = "tut_ex30: Table with dynamic 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_content_dyncb(tbl, "cb_content");
    hpdf_tbl_set_label_dyncb(tbl, "cb_labels");
    hpdf_tbl_set_cell_content_dyncb(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
    // Stroke the table to the page
    hpdf_tbl_stroke(pdf_doc, pdf_page, tbl, xpos, ypos, width, height);
}
TUTEX_MAIN(create_table_ex30, FALSE)
```


Index

- [_HPDFTBL_SET_ERR](#)
hpdtbl.h, [135](#)
 - [_HPDFTBL_SET_ERR_EXTRA](#)
hpdtbl.h, [135](#)
- background
 - text_style, [82](#)
- bootstrap.sh, [92](#)
- bottom_vmargin_factor
 - hpdtbl, [59](#)
 - hpdtbl_theme, [77](#)
- canvas_cb
 - hpdtbl, [59](#)
 - hpdtbl_cell, [67](#)
 - hpdtbl_cell_spec, [70](#)
- cell_spec
 - hpdtbl_spec, [74](#)
- cells
 - hpdtbl, [60](#)
- CENTER
 - hpdtbl.h, [143](#)
- chktbl
 - hpdtbl.c, [100](#)
 - hpdtbl.h, [143](#)
- col
 - hpdtbl_cell_spec, [70](#)
- col_width_percent
 - hpdtbl, [60](#)
- color
 - grid_style, [57](#)
 - text_style, [82](#)
- cols
 - hpdtbl, [60](#)
 - hpdtbl_spec, [74](#)
- colspan
 - hpdtbl_cell, [67](#)
 - hpdtbl_cell_spec, [71](#)
- config.h, [95](#)
- content
 - hpdtbl_cell, [67](#)
- content_cb
 - hpdtbl, [60](#)
 - hpdtbl_cell, [67](#)
 - hpdtbl_cell_spec, [71](#)
 - hpdtbl_spec, [74](#)
- content_style
 - hpdtbl, [60](#)
 - hpdtbl_cell, [67](#)
 - hpdtbl_theme, [78](#)
- content_style_cb
 - hpdtbl, [61](#)
- dash_ptn
 - line_dash_style, [81](#)
- dbgbl.sh, [93](#)
- DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR
 - hpdtbl.h, [137](#)
- delta_x
 - hpdtbl_cell, [68](#)
- delta_y
 - hpdtbl_cell, [68](#)
- docupload.sh.in, [93](#)
 - GITHUB_USER, [94](#)
 - PDFFILE_COPY, [94](#)
- errcode
 - hpdtbl_errcode_entry, [73](#)
- errstr
 - hpdtbl_errcode_entry, [73](#)
- FALSE
 - hpdtbl_widget.c, [212](#)
- font
 - text_style, [83](#)
- fsize
 - text_style, [83](#)
- GITHUB_USER
 - docupload.sh.in, [94](#)
- grid_style, [57](#)
 - color, [57](#)
 - line_dashstyle, [57](#)
 - width, [58](#)
- halign
 - text_style, [83](#)
- header_style
 - hpdtbl, [61](#)
 - hpdtbl_theme, [78](#)
- height
 - hpdtbl, [61](#)
 - hpdtbl_cell, [68](#)
 - hpdtbl_spec, [74](#)
- HPDF_RoundedCornerRectangle
 - hpdtbl.c, [100](#)
 - hpdtbl.h, [143](#)
- hpdf_text_style_t
 - hpdtbl.h, [137](#)
- hpdtbl, [58](#)
 - bottom_vmargin_factor, [59](#)

- canvas_cb, 59
- cells, 60
- col_width_percent, 60
- cols, 60
- content_cb, 60
- content_style, 60
- content_style_cb, 61
- header_style, 61
- height, 61
- inner_hgrid, 61
- inner_tgrid, 61
- inner_vgrid, 62
- label_cb, 62
- label_style, 62
- minheight, 62
- outer_grid, 62
- pdf_doc, 63
- pdf_page, 63
- posx, 63
- posy, 63
- rows, 63
- tag, 63
- title_style, 64
- title_txt, 64
- use_cell_labels, 64
- use_header_row, 64
- use_label_grid_style, 64
- use_zebra, 65
- width, 65
- zebra_color1, 65
- zebra_color2, 65
- zebra_phase, 66
- hpdfctl.c, 96
 - chkctl, 100
 - HPDF_RoundedCornerRectangle, 100
 - hpdfctl_clear_spanning, 101
 - hpdfctl_create, 101
 - hpdfctl_create_title, 102
 - hpdfctl_default_table_error_handler, 102
 - hpdfctl_destroy, 103
 - hpdfctl_encoding_text_out, 103
 - hpdfctl_get_anchor_top_left, 104
 - hpdfctl_get_errstr, 104
 - hpdfctl_get_last_auto_height, 105
 - hpdfctl_get_last_err_file, 105
 - hpdfctl_get_last_errcode, 106
 - hpdfctl_set_anchor_top_left, 106
 - hpdfctl_set_background, 106
 - hpdfctl_set_bottom_vmargin_factor, 107
 - hpdfctl_set_cell, 107
 - hpdfctl_set_cell_content_style, 108
 - hpdfctl_set_cellspan, 108
 - hpdfctl_set_col_content_style, 109
 - hpdfctl_set_colwidth_percent, 110
 - hpdfctl_set_content, 110
 - hpdfctl_set_content_style, 111
 - hpdfctl_set_errhandler, 112
 - hpdfctl_set_header_halign, 112
 - hpdfctl_set_header_style, 113
 - hpdfctl_set_inner_grid_style, 114
 - hpdfctl_set_inner_hgrid_style, 114
 - hpdfctl_set_inner_tgrid_style, 115
 - hpdfctl_set_inner_vgrid_style, 115
 - hpdfctl_set_label_style, 116
 - hpdfctl_set_labels, 117
 - hpdfctl_set_line_dash, 117
 - hpdfctl_set_min_rowheight, 118
 - hpdfctl_set_outer_grid_style, 118
 - hpdfctl_set_row_content_style, 119
 - hpdfctl_set_tag, 120
 - hpdfctl_set_text_encoding, 120
 - hpdfctl_set_title, 120
 - hpdfctl_set_title_halign, 121
 - hpdfctl_set_title_style, 122
 - hpdfctl_set_zebra, 122
 - hpdfctl_set_zebra_color, 123
 - hpdfctl_stroke, 123
 - hpdfctl_stroke_from_data, 124
 - hpdfctl_stroke_pdfdoc, 125
 - hpdfctl_use_header, 125
 - hpdfctl_use_labelgrid, 126
 - hpdfctl_use_labels, 127
- hpdfctl.h, 127, 185
 - _HPDFCTL_SET_ERR, 135
 - _HPDFCTL_SET_ERR_EXTRA, 135
 - CENTER, 143
 - chkctl, 143
 - DEFAULT_AUTO_VBOTTOM_MARGIN_FACTOR, 137
 - HPDF_RoundedCornerRectangle, 143
 - hpdf_text_style_t, 137
 - hpdfctl_apply_theme, 144
 - hpdfctl_callback_t, 138
 - hpdfctl_canvas_callback_t, 138
 - hpdfctl_cell_spec_t, 138
 - hpdfctl_cell_t, 138
 - hpdfctl_clear_spanning, 144
 - hpdfctl_cm2dpi, 137
 - hpdfctl_content_callback_t, 139
 - hpdfctl_content_style_callback_t, 139
 - hpdfctl_create, 145
 - hpdfctl_create_title, 145
 - hpdfctl_dashstyle, 141
 - hpdfctl_default_table_error_handler, 146
 - hpdfctl_destroy, 146
 - hpdfctl_destroy_theme, 147
 - hpdfctl_encoding_text_out, 147
 - hpdfctl_error_handler_t, 139
 - hpdfctl_get_anchor_top_left, 148
 - hpdfctl_get_default_theme, 148
 - hpdfctl_get_errstr, 148
 - hpdfctl_get_last_auto_height, 149
 - hpdfctl_get_last_err_file, 149
 - hpdfctl_get_last_errcode, 150
 - hpdfctl_grid_style_t, 139
 - hpdfctl_hpdc_get_errstr, 150

- hpdfctl_line_dashstyle_t, 140
- hpdfctl_set_anchor_top_left, 150
- hpdfctl_set_background, 151
- hpdfctl_set_bottom_vmargin_factor, 151
- hpdfctl_set_canvas_cb, 152
- hpdfctl_set_cell, 152
- hpdfctl_set_cell_canvas_cb, 153
- hpdfctl_set_cell_canvas_dyncb, 153
- hpdfctl_set_cell_content_cb, 154
- hpdfctl_set_cell_content_dyncb, 155
- hpdfctl_set_cell_content_style, 155
- hpdfctl_set_cell_content_style_cb, 156
- hpdfctl_set_cell_content_style_dyncb, 157
- hpdfctl_set_cell_label_cb, 157
- hpdfctl_set_cell_label_dyncb, 158
- hpdfctl_set_cellspan, 159
- hpdfctl_set_col_content_style, 159
- hpdfctl_set_colwidth_percent, 160
- hpdfctl_set_content, 160
- hpdfctl_set_content_cb, 161
- hpdfctl_set_content_dyncb, 162
- hpdfctl_set_content_style, 162
- hpdfctl_set_content_style_cb, 163
- hpdfctl_set_content_style_dyncb, 164
- hpdfctl_set_dlhandle, 164
- hpdfctl_set_errhandler, 165
- hpdfctl_set_header_halign, 165
- hpdfctl_set_header_style, 166
- hpdfctl_set_inner_grid_style, 166
- hpdfctl_set_inner_hgrid_style, 167
- hpdfctl_set_inner_tgrid_style, 167
- hpdfctl_set_inner_vgrid_style, 168
- hpdfctl_set_label_cb, 169
- hpdfctl_set_label_dyncb, 169
- hpdfctl_set_label_style, 170
- hpdfctl_set_labels, 171
- hpdfctl_set_min_rowheight, 171
- hpdfctl_set_outer_grid_style, 172
- hpdfctl_set_row_content_style, 172
- hpdfctl_set_tag, 173
- hpdfctl_set_text_encoding, 173
- hpdfctl_set_title, 174
- hpdfctl_set_title_halign, 174
- hpdfctl_set_title_style, 175
- hpdfctl_set_zebra, 175
- hpdfctl_set_zebra_color, 176
- hpdfctl_spec_t, 140
- hpdfctl_stroke, 176
- hpdfctl_stroke_from_data, 177
- hpdfctl_stroke_grid, 178
- hpdfctl_stroke_pdfdoc, 178
- hpdfctl_t, 140
- hpdfctl_table_widget_letter_buttons, 179
- hpdfctl_text_align, 141
- hpdfctl_text_align_t, 140
- hpdfctl_theme_t, 140
- hpdfctl_use_header, 179
- hpdfctl_use_labelgrid, 180
- hpdfctl_use_labels, 181
- hpdfctl_widget_hbar, 181
- hpdfctl_widget_segment_hbar, 182
- hpdfctl_widget_slide_button, 183
- hpdfctl_widget_strength_meter, 183
- LEFT, 143
- LINE_DASH1, 141
- LINE_DASH2, 141
- LINE_DASH3, 141
- LINE_DASH4, 141
- LINE_DASH5, 141
- LINE_DASHDOT1, 141
- LINE_DASHDOT2, 141
- LINE_DOT1, 141
- LINE_DOT2, 141
- LINE_DOT3, 141
- LINE_DOT4, 141
- LINE_SOLID, 141
- RIGHT, 143
- hpdfctl_apply_theme
 - hpdfctl.h, 144
 - hpdfctl_theme.c, 209
- hpdfctl_callback.c, 192
 - hpdfctl_set_canvas_cb, 193
 - hpdfctl_set_cell_canvas_cb, 194
 - hpdfctl_set_cell_canvas_dyncb, 194
 - hpdfctl_set_cell_content_cb, 195
 - hpdfctl_set_cell_content_dyncb, 196
 - hpdfctl_set_cell_content_style_cb, 196
 - hpdfctl_set_cell_content_style_dyncb, 197
 - hpdfctl_set_cell_label_cb, 198
 - hpdfctl_set_cell_label_dyncb, 198
 - hpdfctl_set_content_cb, 200
 - hpdfctl_set_content_dyncb, 200
 - hpdfctl_set_content_style_cb, 201
 - hpdfctl_set_content_style_dyncb, 202
 - hpdfctl_set_dlhandle, 202
 - hpdfctl_set_label_cb, 203
 - hpdfctl_set_label_dyncb, 203
- hpdfctl_callback_t
 - hpdfctl.h, 138
- hpdfctl_canvas_callback_t
 - hpdfctl.h, 138
- hpdfctl_cell, 66
 - canvas_cb, 67
 - colspan, 67
 - content, 67
 - content_cb, 67
 - content_style, 67
 - delta_x, 68
 - delta_y, 68
 - height, 68
 - label, 68
 - label_cb, 68
 - parent_cell, 68
 - rowspan, 69
 - style_cb, 69
 - textwidth, 69

- width, 69
- hpdfctl_cell_spec, 70
 - canvas_cb, 70
 - col, 70
 - colspan, 71
 - content_cb, 71
 - label, 71
 - label_cb, 71
 - row, 71
 - rowspan, 72
 - style_cb, 72
- hpdfctl_cell_spec_t
 - hpdfctl.h, 138
- hpdfctl_cell_t
 - hpdfctl.h, 138
- hpdfctl_clear_spanning
 - hpdfctl.c, 101
 - hpdfctl.h, 144
- hpdfctl_cm2dpi
 - hpdfctl.h, 137
- hpdfctl_content_callback_t
 - hpdfctl.h, 139
- hpdfctl_content_style_callback_t
 - hpdfctl.h, 139
- hpdfctl_create
 - hpdfctl.c, 101
 - hpdfctl.h, 145
- hpdfctl_create_title
 - hpdfctl.c, 102
 - hpdfctl.h, 145
- hpdfctl_dashstyle
 - hpdfctl.h, 141
- HPDFTBL_DEFAULT_CONTENT_STYLE
 - hpdfctl_theme.c, 207
- HPDFTBL_DEFAULT_HEADER_STYLE
 - hpdfctl_theme.c, 207
- HPDFTBL_DEFAULT_INNER_HGRID_STYLE
 - hpdfctl_theme.c, 208
- HPDFTBL_DEFAULT_INNER_VGRID_STYLE
 - hpdfctl_theme.c, 208
- HPDFTBL_DEFAULT_LABEL_STYLE
 - hpdfctl_theme.c, 208
- HPDFTBL_DEFAULT_OUTER_GRID_STYLE
 - hpdfctl_theme.c, 208
- hpdfctl_default_table_error_handler
 - hpdfctl.c, 102
 - hpdfctl.h, 146
- HPDFTBL_DEFAULT_ZEBRA_COLOR1
 - hpdfctl_theme.c, 209
- HPDFTBL_DEFAULT_ZEBRA_COLOR2
 - hpdfctl_theme.c, 209
- hpdfctl_destroy
 - hpdfctl.c, 103
 - hpdfctl.h, 146
- hpdfctl_destroy_theme
 - hpdfctl.h, 147
 - hpdfctl_theme.c, 210
- hpdfctl_encoding_text_out
 - hpdfctl.c, 103
- hpdfctl.h, 147
- hpdfctl_errcode_entry, 72
 - errcode, 73
 - errstr, 73
- hpdfctl_error_handler_t
 - hpdfctl.h, 139
- hpdfctl_errstr.c, 204
 - hpdfctl_hpdc_get_errstr, 204
- hpdfctl_get_anchor_top_left
 - hpdfctl.c, 104
 - hpdfctl.h, 148
- hpdfctl_get_default_theme
 - hpdfctl.h, 148
 - hpdfctl_theme.c, 210
- hpdfctl_get_errstr
 - hpdfctl.c, 104
 - hpdfctl.h, 148
- hpdfctl_get_last_auto_height
 - hpdfctl.c, 105
 - hpdfctl.h, 149
- hpdfctl_get_last_err_file
 - hpdfctl.c, 105
 - hpdfctl.h, 149
- hpdfctl_get_last_errcode
 - hpdfctl.c, 106
 - hpdfctl.h, 150
- hpdfctl_grid.c, 205
 - hpdfctl_stroke_grid, 205
- hpdfctl_grid_style_t
 - hpdfctl.h, 139
- hpdfctl_hpdc_get_errstr
 - hpdfctl.h, 150
 - hpdfctl_errstr.c, 204
- hpdfctl_line_dashstyle_t
 - hpdfctl.h, 140
- hpdfctl_set_anchor_top_left
 - hpdfctl.c, 106
 - hpdfctl.h, 150
- hpdfctl_set_background
 - hpdfctl.c, 106
 - hpdfctl.h, 151
- hpdfctl_set_bottom_vmargin_factor
 - hpdfctl.c, 107
 - hpdfctl.h, 151
- hpdfctl_set_canvas_cb
 - hpdfctl.h, 152
 - hpdfctl_callback.c, 193
- hpdfctl_set_cell
 - hpdfctl.c, 107
 - hpdfctl.h, 152
- hpdfctl_set_cell_canvas_cb
 - hpdfctl.h, 153
 - hpdfctl_callback.c, 194
- hpdfctl_set_cell_canvas_dyncb
 - hpdfctl.h, 153
 - hpdfctl_callback.c, 194
- hpdfctl_set_cell_content_cb

hpdtbl.h, 154
hpdtbl_callback.c, 195
hpdtbl_set_cell_content_dyncb
 hpdtbl.h, 155
 hpdtbl_callback.c, 196
hpdtbl_set_cell_content_style
 hpdtbl.c, 108
 hpdtbl.h, 155
hpdtbl_set_cell_content_style_cb
 hpdtbl.h, 156
 hpdtbl_callback.c, 196
hpdtbl_set_cell_content_style_dyncb
 hpdtbl.h, 157
 hpdtbl_callback.c, 197
hpdtbl_set_cell_label_cb
 hpdtbl.h, 157
 hpdtbl_callback.c, 198
hpdtbl_set_cell_label_dyncb
 hpdtbl.h, 158
 hpdtbl_callback.c, 198
hpdtbl_set_cellspan
 hpdtbl.c, 108
 hpdtbl.h, 159
hpdtbl_set_col_content_style
 hpdtbl.c, 109
 hpdtbl.h, 159
hpdtbl_set_colwidth_percent
 hpdtbl.c, 110
 hpdtbl.h, 160
hpdtbl_set_content
 hpdtbl.c, 110
 hpdtbl.h, 160
hpdtbl_set_content_cb
 hpdtbl.h, 161
 hpdtbl_callback.c, 200
hpdtbl_set_content_dyncb
 hpdtbl.h, 162
 hpdtbl_callback.c, 200
hpdtbl_set_content_style
 hpdtbl.c, 111
 hpdtbl.h, 162
hpdtbl_set_content_style_cb
 hpdtbl.h, 163
 hpdtbl_callback.c, 201
hpdtbl_set_content_style_dyncb
 hpdtbl.h, 164
 hpdtbl_callback.c, 202
hpdtbl_set_dlhandle
 hpdtbl.h, 164
 hpdtbl_callback.c, 202
hpdtbl_set_errhandler
 hpdtbl.c, 112
 hpdtbl.h, 165
hpdtbl_set_header_halign
 hpdtbl.c, 112
 hpdtbl.h, 165
hpdtbl_set_header_style
 hpdtbl.c, 113
 hpdtbl.h, 166
hpdtbl_set_inner_grid_style
 hpdtbl.c, 114
 hpdtbl.h, 166
hpdtbl_set_inner_hgrid_style
 hpdtbl.c, 114
 hpdtbl.h, 167
hpdtbl_set_inner_tgrid_style
 hpdtbl.c, 115
 hpdtbl.h, 167
hpdtbl_set_inner_vgrid_style
 hpdtbl.c, 115
 hpdtbl.h, 168
hpdtbl_set_label_cb
 hpdtbl.h, 169
 hpdtbl_callback.c, 203
hpdtbl_set_label_dyncb
 hpdtbl.h, 169
 hpdtbl_callback.c, 203
hpdtbl_set_label_style
 hpdtbl.c, 116
 hpdtbl.h, 170
hpdtbl_set_labels
 hpdtbl.c, 117
 hpdtbl.h, 171
hpdtbl_set_line_dash
 hpdtbl.c, 117
hpdtbl_set_min_rowheight
 hpdtbl.c, 118
 hpdtbl.h, 171
hpdtbl_set_outer_grid_style
 hpdtbl.c, 118
 hpdtbl.h, 172
hpdtbl_set_row_content_style
 hpdtbl.c, 119
 hpdtbl.h, 172
hpdtbl_set_tag
 hpdtbl.c, 120
 hpdtbl.h, 173
hpdtbl_set_text_encoding
 hpdtbl.c, 120
 hpdtbl.h, 173
hpdtbl_set_title
 hpdtbl.c, 120
 hpdtbl.h, 174
hpdtbl_set_title_halign
 hpdtbl.c, 121
 hpdtbl.h, 174
hpdtbl_set_title_style
 hpdtbl.c, 122
 hpdtbl.h, 175
hpdtbl_set_zebra
 hpdtbl.c, 122
 hpdtbl.h, 175
hpdtbl_set_zebra_color
 hpdtbl.c, 123
 hpdtbl.h, 176
hpdtbl_spec, 73

- cell_spec, [74](#)
- cols, [74](#)
- content_cb, [74](#)
- height, [74](#)
- label_cb, [74](#)
- post_cb, [75](#)
- rows, [75](#)
- style_cb, [75](#)
- title, [75](#)
- use_header, [75](#)
- use_labelgrid, [76](#)
- use_labels, [76](#)
- width, [76](#)
- xpos, [76](#)
- ypos, [76](#)
- hpdfctl_spec_t
 - hpdfctl.h, [140](#)
- hpdfctl_stroke
 - hpdfctl.c, [123](#)
 - hpdfctl.h, [176](#)
- hpdfctl_stroke_from_data
 - hpdfctl.c, [124](#)
 - hpdfctl.h, [177](#)
- hpdfctl_stroke_grid
 - hpdfctl.h, [178](#)
 - hpdfctl_grid.c, [205](#)
- hpdfctl_stroke_pdfdoc
 - hpdfctl.c, [125](#)
 - hpdfctl.h, [178](#)
- hpdfctl_t
 - hpdfctl.h, [140](#)
- hpdfctl_table_widget_letter_buttons
 - hpdfctl.h, [179](#)
 - hpdfctl_widget.c, [212](#)
- hpdfctl_text_align
 - hpdfctl.h, [141](#)
- hpdfctl_text_align_t
 - hpdfctl.h, [140](#)
- hpdfctl_theme, [77](#)
 - bottom_vmargin_factor, [77](#)
 - content_style, [78](#)
 - header_style, [78](#)
 - inner_hborder, [78](#)
 - inner_tborder, [78](#)
 - inner_vborder, [78](#)
 - label_style, [79](#)
 - outer_border, [79](#)
 - title_style, [79](#)
 - use_header_row, [79](#)
 - use_label_grid_style, [79](#)
 - use_labels, [80](#)
 - use_zebra, [80](#)
 - zebra_color1, [80](#)
 - zebra_color2, [80](#)
 - zebra_phase, [80](#)
- hpdfctl_theme.c, [206](#)
 - hpdfctl_apply_theme, [209](#)
 - HPDFTBL_DEFAULT_CONTENT_STYLE, [207](#)
 - HPDFTBL_DEFAULT_HEADER_STYLE, [207](#)
 - HPDFTBL_DEFAULT_INNER_HGRID_STYLE, [208](#)
 - HPDFTBL_DEFAULT_INNER_VGRID_STYLE, [208](#)
 - HPDFTBL_DEFAULT_LABEL_STYLE, [208](#)
 - HPDFTBL_DEFAULT_OUTER_GRID_STYLE, [208](#)
 - HPDFTBL_DEFAULT_ZEBRA_COLOR1, [209](#)
 - HPDFTBL_DEFAULT_ZEBRA_COLOR2, [209](#)
 - hpdfctl_destroy_theme, [210](#)
 - hpdfctl_get_default_theme, [210](#)
- hpdfctl_theme_t
 - hpdfctl.h, [140](#)
- hpdfctl_use_header
 - hpdfctl.c, [125](#)
 - hpdfctl.h, [179](#)
- hpdfctl_use_labelgrid
 - hpdfctl.c, [126](#)
 - hpdfctl.h, [180](#)
- hpdfctl_use_labels
 - hpdfctl.c, [127](#)
 - hpdfctl.h, [181](#)
- hpdfctl_widget.c, [211](#)
 - FALSE, [212](#)
 - hpdfctl_table_widget_letter_buttons, [212](#)
 - hpdfctl_widget_hbar, [213](#)
 - hpdfctl_widget_segment_hbar, [213](#)
 - hpdfctl_widget_slide_button, [214](#)
 - hpdfctl_widget_strength_meter, [215](#)
 - TRUE, [212](#)
- hpdfctl_widget_hbar
 - hpdfctl.h, [181](#)
 - hpdfctl_widget.c, [213](#)
- hpdfctl_widget_segment_hbar
 - hpdfctl.h, [182](#)
 - hpdfctl_widget.c, [213](#)
- hpdfctl_widget_slide_button
 - hpdfctl.h, [183](#)
 - hpdfctl_widget.c, [214](#)
- hpdfctl_widget_strength_meter
 - hpdfctl.h, [183](#)
 - hpdfctl_widget.c, [215](#)
- inner_hborder
 - hpdfctl_theme, [78](#)
- inner_hgrid
 - hpdfctl, [61](#)
- inner_tborder
 - hpdfctl_theme, [78](#)
- inner_tgrid
 - hpdfctl, [61](#)
- inner_vborder
 - hpdfctl_theme, [78](#)
- inner_vgrid
 - hpdfctl, [62](#)
- label
 - hpdfctl_cell, [68](#)
 - hpdfctl_cell_spec, [71](#)

label_cb
 hpdftbl, 62
 hpdftbl_cell, 68
 hpdftbl_cell_spec, 71
 hpdftbl_spec, 74
label_style
 hpdftbl, 62
 hpdftbl_theme, 79
LEFT
 hpdftbl.h, 143
LINE_DASH1
 hpdftbl.h, 141
LINE_DASH2
 hpdftbl.h, 141
LINE_DASH3
 hpdftbl.h, 141
LINE_DASH4
 hpdftbl.h, 141
LINE_DASH5
 hpdftbl.h, 141
line_dash_style, 81
 dash_ptn, 81
 num, 81
LINE_DASHDOT1
 hpdftbl.h, 141
LINE_DASHDOT2
 hpdftbl.h, 141
line_dashstyle
 grid_style, 57
LINE_DOT1
 hpdftbl.h, 141
LINE_DOT2
 hpdftbl.h, 141
LINE_DOT3
 hpdftbl.h, 141
LINE_DOT4
 hpdftbl.h, 141
LINE_SOLID
 hpdftbl.h, 141

minheight
 hpdftbl, 62

num
 line_dash_style, 81

outer_border
 hpdftbl_theme, 79
outer_grid
 hpdftbl, 62

parent_cell
 hpdftbl_cell, 68
pdf_doc
 hpdftbl, 63
pdf_page
 hpdftbl, 63
PDFFILE_COPY
 docupload.sh.in, 94

post_cb
 hpdftbl_spec, 75
posix
 hpdftbl, 63
posy
 hpdftbl, 63

RIGHT
 hpdftbl.h, 143
row
 hpdftbl_cell_spec, 71
rows
 hpdftbl, 63
 hpdftbl_spec, 75
rowspan
 hpdftbl_cell, 69
 hpdftbl_cell_spec, 72
run_as_unit_test
 unit_test.inc.h, 90

setup_dummy_content
 unit_test.inc.h, 87
setup_dummy_content_label
 unit_test.inc.h, 87
setup_filename
 unit_test.inc.h, 88
setup_hpdf
 unit_test.inc.h, 88
stdbld.sh, 94
stroke_to_file
 unit_test.inc.h, 89
style_cb
 hpdftbl_cell, 69
 hpdftbl_cell_spec, 72
 hpdftbl_spec, 75

tag
 hpdftbl, 63
text_style, 82
 background, 82
 color, 82
 font, 83
 fsize, 83
 halign, 83
textwidth
 hpdftbl_cell, 69
title
 hpdftbl_spec, 75
title_style
 hpdftbl, 64
 hpdftbl_theme, 79
title_txt
 hpdftbl, 64
TRUE
 hpdftbl_widget.c, 212
TUTEX_MAIN
 unit_test.inc.h, 86
unit_test.inc.h, 85, 90

- run_as_unit_test, [90](#)
- setup_dummy_content, [87](#)
- setup_dummy_content_label, [87](#)
- setup_filename, [88](#)
- setup_hpdf, [88](#)
- stroke_to_file, [89](#)
- TUTEX_MAIN, [86](#)
- use_cell_labels
 - hpdfitbl, [64](#)
- use_header
 - hpdfitbl_spec, [75](#)
- use_header_row
 - hpdfitbl, [64](#)
 - hpdfitbl_theme, [79](#)
- use_label_grid_style
 - hpdfitbl, [64](#)
 - hpdfitbl_theme, [79](#)
- use_labelgrid
 - hpdfitbl_spec, [76](#)
- use_labels
 - hpdfitbl_spec, [76](#)
 - hpdfitbl_theme, [80](#)
- use_zebra
 - hpdfitbl, [65](#)
 - hpdfitbl_theme, [80](#)
- width
 - grid_style, [58](#)
 - hpdfitbl, [65](#)
 - hpdfitbl_cell, [69](#)
 - hpdfitbl_spec, [76](#)
- xpos
 - hpdfitbl_spec, [76](#)
- ypos
 - hpdfitbl_spec, [76](#)
- zebra_color1
 - hpdfitbl, [65](#)
 - hpdfitbl_theme, [80](#)
- zebra_color2
 - hpdfitbl, [65](#)
 - hpdfitbl_theme, [80](#)
- zebra_phase
 - hpdfitbl, [66](#)
 - hpdfitbl_theme, [80](#)