

GDS-Render

v1.0

Generated by Doxygen 1.8.16

1 Main Page	1
2 Compilation	3
2.1 Preface	3
2.2 Dependencies	3
2.2.1 Program Dependencies	3
2.2.2 Compilation Dependencies	3
2.3 Compilation Instructions	4
2.3.1 General Linux Build Instruction	4
2.3.2 Archlinux Package	4
2.3.3 Warnings	4
3 Layer Mapping File Specification	5
4 Usage	7
4.1 Command Line Interface	7
4.2 Graphical User Interface	7
5 Version Number	9
5.1 Main Versioning Scheme	9
5.1.1 Release Candidates	9
5.1.2 Patch Levels	9
5.2 Git Based Version Number	9
6 GNU GENERAL PUBLIC LICENSE	11
7 GDS-Render Readme	17
8 Module Index	19
8.1 Modules	19
9 Data Structure Index	21
9.1 Data Structures	21
10 File Index	23
10.1 File List	23
11 Module Documentation	25
11.1 Cairo Renderer	25
11.1.1 Detailed Description	26
11.1.2 Macro Definition Documentation	26
11.1.2.1 MAX_LAYERS	26
11.1.3 Function Documentation	26
11.1.3.1 apply_inherited_transform_to_all_layers()	26
11.1.3.2 cairo_render_cell_to_vector_file()	27
11.1.3.3 render_cell()	28

11.1.3.4 revert_inherited_transform()	28
11.2 Command Line Interface	30
11.2.1 Detailed Description	30
11.2.2 Function Documentation	30
11.2.2.1 command_line_convert_gds()	30
11.2.2.2 delete_layer_info_with_name()	31
11.3 External Shared Object Renderer	33
11.3.1 Detailed Description	33
11.3.2 Macro Definition Documentation	33
11.3.2.1 EXTERNAL_LIBRARY_FUNCTION	33
11.3.3 Function Documentation	33
11.3.3.1 external_renderer_render_cell()	33
11.4 Geometric Helper Functions	35
11.4.1 Detailed Description	36
11.4.2 Macro Definition Documentation	36
11.4.2.1 ABS_DBL [1/2]	36
11.4.2.2 ABS_DBL [2/2]	36
11.4.2.3 DEG2RAD	36
11.4.2.4 MAX	36
11.4.2.5 MIN	37
11.4.3 Typedef Documentation	37
11.4.3.1 conv_generic_to_vector_2d_t	37
11.4.4 Function Documentation	37
11.4.4.1 bounding_box_apply_transform()	37
11.4.4.2 bounding_box_calculate_path_box()	38
11.4.4.3 bounding_box_calculate_polygon()	39
11.4.4.4 bounding_box_prepare_empty()	39
11.4.4.5 bounding_box_update_box()	40
11.4.4.6 bounding_box_update_point()	40
11.4.4.7 calculate_cell_bounding_box()	40
11.4.4.8 calculate_path_miter_points()	41
11.4.4.9 convert_gds_point_to_2d_vector()	42
11.4.4.10 update_box_with_gfx()	42
11.4.4.11 vector_2d_abs()	43
11.4.4.12 vector_2d_add()	44
11.4.4.13 vector_2d_alloc()	44
11.4.4.14 vector_2d_calculate_angle_between()	45
11.4.4.15 vector_2d_copy()	45
11.4.4.16 vector_2d_free()	46
11.4.4.17 vector_2d_normalize()	46
11.4.4.18 vector_2d_rotate()	47
11.4.4.19 vector_2d_scalar_multiply()	47

11.4.4.20	vector_2d_scale()	48
11.4.4.21	vector_2d_subtract()	48
11.5	Graphical User Interface	49
11.5.1	Detailed Description	50
11.5.2	Macro Definition Documentation	50
11.5.2.1	RENDERER_TYPE_GUI	50
11.5.3	Enumeration Type Documentation	51
11.5.3.1	cell_store_columns	51
11.5.3.2	gds_render_gui_signal_sig_ids	51
11.5.4	Function Documentation	51
11.5.4.1	cell_selection_changed()	51
11.5.4.2	cell_store_filter_visible_func()	52
11.5.4.3	cell_tree_view_activated()	53
11.5.4.4	change_filter()	54
11.5.4.5	G_DECLARE_FINAL_TYPE()	54
11.5.4.6	gds_render_gui_class_init()	54
11.5.4.7	gds_render_gui_dispose()	55
11.5.4.8	gds_render_gui_get_main_window()	55
11.5.4.9	gds_render_gui_init()	56
11.5.4.10	gds_render_gui_new()	56
11.5.4.11	generate_string_from_date()	57
11.5.4.12	on_convert_clicked()	57
11.5.4.13	on_load_gds()	58
11.5.4.14	on_window_close()	60
11.5.4.15	setup_cell_selector()	60
11.5.4.16	sort_down_callback()	61
11.5.4.17	sort_up_callback()	62
11.5.4.18	tree_sel_func()	63
11.5.5	Variable Documentation	63
11.5.5.1	gds_render_gui_signals	63
11.6	LaTeX/TikZ Renderer	64
11.6.1	Detailed Description	64
11.6.2	Macro Definition Documentation	64
11.6.2.1	LATEX_LINE_BUFFER_KB	64
11.6.2.2	WRITEOUT_BUFFER	65
11.6.3	Function Documentation	65
11.6.3.1	generate_graphics()	65
11.6.3.2	latex_render_cell_to_code()	66
11.6.3.3	render_cell()	67
11.6.3.4	write_layer_definitions()	67
11.6.3.5	write_layer_env()	69
11.7	LayerSelector Object	71

11.7.1 Detailed Description	72
11.7.2 Macro Definition Documentation	72
11.7.2.1 TYPE_LAYER_SELECTOR	72
11.7.3 Enumeration Type Documentation	73
11.7.3.1 layer_selector_sort_algo	73
11.7.4 Function Documentation	73
11.7.4.1 G_DECLARE_FINAL_TYPE()	73
11.7.4.2 layer_selector_analyze_cell_layers()	73
11.7.4.3 layer_selector_check_if_layer_widget_exists()	74
11.7.4.4 layer_selector_class_init()	75
11.7.4.5 layer_selector_clear_widgets()	75
11.7.4.6 layer_selector_dispose()	76
11.7.4.7 layer_selector_drag_data_received()	76
11.7.4.8 layer_selector_drag_leave()	77
11.7.4.9 layer_selector_drag_motion()	77
11.7.4.10 layer_selector_export_rendered_layer_info()	78
11.7.4.11 layer_selector_find_layer_element_in_list()	79
11.7.4.12 layer_selector_force_sort()	80
11.7.4.13 layer_selector_generate_layer_widgets()	80
11.7.4.14 layer_selector_get_last_row()	81
11.7.4.15 layer_selector_get_row_after()	82
11.7.4.16 layer_selector_get_row_before()	82
11.7.4.17 layer_selector_init()	82
11.7.4.18 layer_selector_load_layer_mapping_from_file()	82
11.7.4.19 layer_selector_load_mapping_clicked()	83
11.7.4.20 layer_selector_new()	84
11.7.4.21 layer_selector_save_layer_mapping_data()	85
11.7.4.22 layer_selector_save_mapping_clicked()	86
11.7.4.23 layer_selector_set_load_mapping_button()	87
11.7.4.24 layer_selector_set_save_mapping_button()	88
11.7.4.25 layer_selector_setup_dnd()	88
11.7.4.26 layer_selector_sort_func()	89
11.7.4.27 sel_layer_element_drag_begin()	90
11.7.4.28 sel_layer_element_drag_data_get()	91
11.7.4.29 sel_layer_element_drag_end()	91
11.7.4.30 sel_layer_element_setup_dnd_callbacks()	91
11.7.5 Variable Documentation	92
11.7.5.1 dnd_additional_css	92
11.8 LibCellRenderer GObject	93
11.8.1 Detailed Description	94
11.8.2 Macro Definition Documentation	94
11.8.2.1 LIB_CELL_RENDERER_ERROR_ERR	94

11.8.2.2 LIB_CELL_RENDERER_ERROR_WARN	94
11.8.2.3 TYPE_LIB_CELL_RENDERER	94
11.8.3 Typedef Documentation	94
11.8.3.1 LibCellRenderer	95
11.8.4 Enumeration Type Documentation	95
11.8.4.1 anonymous enum	95
11.8.5 Function Documentation	95
11.8.5.1 convert_error_level_to_color()	95
11.8.5.2 lib_cell_renderer_class_init()	96
11.8.5.3 lib_cell_renderer_constructed()	96
11.8.5.4 lib_cell_renderer_get_property()	96
11.8.5.5 lib_cell_renderer_get_type()	97
11.8.5.6 lib_cell_renderer_init()	97
11.8.5.7 lib_cell_renderer_new()	97
11.8.5.8 lib_cell_renderer_set_property()	98
11.8.6 Variable Documentation	98
11.8.6.1 properties	98
11.9 Output Renderers	99
11.9.1 Detailed Description	99
11.10 Custom GTK Widgets	100
11.10.1 Detailed Description	100
11.11 GDS-Utilities	101
11.11.1 Detailed Description	103
11.11.2 Macro Definition Documentation	103
11.11.2.1 CELL_NAME_MAX	103
11.11.2.2 GDS_DEFAULT_UNITS	103
11.11.2.3 GDS_ERROR	103
11.11.2.4 GDS_INF	104
11.11.2.5 GDS_PRINT_DEBUG_INFOS	104
11.11.2.6 GDS_WARN	104
11.11.2.7 MAX	104
11.11.2.8 MIN	104
11.11.3 Enumeration Type Documentation	104
11.11.3.1 anonymous enum	104
11.11.3.2 gds_record	105
11.11.3.3 graphics_type	105
11.11.3.4 path_type	106
11.11.4 Function Documentation	106
11.11.4.1 append_cell()	106
11.11.4.2 append_cell_ref()	107
11.11.4.3 append_graphics()	108
11.11.4.4 append_library()	108

11.11.4.5	append_vertex()	109
11.11.4.6	clear_lib_list()	109
11.11.4.7	delete_cell_element()	110
11.11.4.8	delete_cell_inst_element()	111
11.11.4.9	delete_graphics_obj()	112
11.11.4.10	delete_library_element()	112
11.11.4.11	delete_vertex()	113
11.11.4.12	gds_convert_double()	114
11.11.4.13	gds_convert_signed_int()	114
11.11.4.14	gds_convert_signed_int16()	115
11.11.4.15	gds_convert_unsigned_int16()	115
11.11.4.16	gds_parse_date()	116
11.11.4.17	gds_tree_check_cell_references()	117
11.11.4.18	gds_tree_check_iterate_ref_and_check()	117
11.11.4.19	gds_tree_check_list_contains_cell()	118
11.11.4.20	gds_tree_check_reference_loops()	119
11.11.4.21	name_cell()	120
11.11.4.22	name_cell_ref()	120
11.11.4.23	name_library()	121
11.11.4.24	parse_gds_from_file()	122
11.11.4.25	parse_reference_list()	123
11.11.4.26	scan_cell_reference_dependencies()	123
11.11.4.27	scan_library_references()	124
11.12	Mapping-Parser	126
11.12.1	Detailed Description	126
11.12.2	Function Documentation	126
11.12.2.1	mapping_parser_gen_csv_line()	126
11.12.2.2	mapping_parser_load_line()	127
11.13	Version Number	129
11.13.1	Detailed Description	129
11.13.2	Variable Documentation	129
11.13.2.1	_app_version_string [1/2]	129
11.13.2.2	_app_version_string [2/2]	129
11.14	RendererSettingsDialog	130
11.14.1	Detailed Description	131
11.14.2	Macro Definition Documentation	131
11.14.2.1	RENDERER_TYPE_SETTINGS_DIALOG	131
11.14.3	Enumeration Type Documentation	131
11.14.3.1	anonymous enum	131
11.14.3.2	output_renderer	132
11.14.4	Function Documentation	132
11.14.4.1	convert_number_to_engineering()	132

11.14.4.2	hide_tex_options()	132
11.14.4.3	latex_render_callback()	133
11.14.4.4	renderer_settings_dialog_class_init()	133
11.14.4.5	renderer_settings_dialog_get_property()	134
11.14.4.6	renderer_settings_dialog_get_settings()	134
11.14.4.7	renderer_settings_dialog_init()	135
11.14.4.8	renderer_settings_dialog_new()	135
11.14.4.9	renderer_settings_dialog_set_cell_height()	136
11.14.4.10	renderer_settings_dialog_set_cell_width()	136
11.14.4.11	renderer_settings_dialog_set_database_unit_scale()	137
11.14.4.12	renderer_settings_dialog_set_property()	138
11.14.4.13	renderer_settings_dialog_set_settings()	138
11.14.4.14	renderer_settings_dialog_update_labels()	139
11.14.4.15	scale_value_changed()	140
11.14.4.16	shape_drawer_drawing_callback()	140
11.14.4.17	show_tex_options()	141
11.14.5	Variable Documentation	141
11.14.5.1	properties	141
11.15	LayerElement	142
11.15.1	Detailed Description	143
11.15.2	Macro Definition Documentation	143
11.15.2.1	TYPE_LAYER_ELEMENT	143
11.15.3	Typedef Documentation	143
11.15.3.1	LayerElementPriv	143
11.15.4	Function Documentation	143
11.15.4.1	layer_element_class_init()	143
11.15.4.2	layer_element_constructed()	144
11.15.4.3	layer_element_dispose()	144
11.15.4.4	layer_element_get_color()	144
11.15.4.5	layer_element_get_export()	145
11.15.4.6	layer_element_get_layer()	145
11.15.4.7	layer_element_get_name()	146
11.15.4.8	layer_element_init()	147
11.15.4.9	layer_element_new()	147
11.15.4.10	layer_element_set_color()	147
11.15.4.11	layer_element_set_dnd_callbacks()	148
11.15.4.12	layer_element_set_export()	148
11.15.4.13	layer_element_set_layer()	149
11.15.4.14	layer_element_set_name()	149
12	Data Structure Documentation	151
12.1	gds_cell_checks::_check_internals Struct Reference	151

12.1.1 Detailed Description	151
12.1.2 Field Documentation	151
12.1.2.1 marker	151
12.2 <code>_GdsRenderGui</code> Struct Reference	152
12.2.1 Detailed Description	152
12.2.2 Field Documentation	152
12.2.2.1 <code>cell_search_entry</code>	152
12.2.2.2 <code>cell_tree_store</code>	153
12.2.2.3 <code>cell_tree_view</code>	153
12.2.2.4 <code>convert_button</code>	153
12.2.2.5 <code>gds_libraries</code>	153
12.2.2.6 <code>layer_selector</code>	153
12.2.2.7 <code>main_window</code>	153
12.2.2.8 <code>parent</code>	154
12.2.2.9 <code>render_dialog_settings</code>	154
12.3 <code>_LayerElement</code> Struct Reference	154
12.3.1 Detailed Description	154
12.3.2 Field Documentation	155
12.3.2.1 <code>parent</code>	155
12.3.2.2 <code>priv</code>	155
12.4 <code>_LayerElementPriv</code> Struct Reference	155
12.4.1 Detailed Description	155
12.4.2 Field Documentation	155
12.4.2.1 <code>color</code>	156
12.4.2.2 <code>event_handle</code>	156
12.4.2.3 <code>export</code>	156
12.4.2.4 <code>layer</code>	156
12.4.2.5 <code>layer_num</code>	156
12.4.2.6 <code>name</code>	156
12.5 <code>_LayerSelector</code> Struct Reference	157
12.5.1 Detailed Description	157
12.5.2 Field Documentation	157
12.5.2.1 <code>associated_load_button</code>	157
12.5.2.2 <code>associated_save_button</code>	157
12.5.2.3 <code>dnd_target</code>	157
12.5.2.4 <code>dummy</code>	158
12.5.2.5 <code>list_box</code>	158
12.5.2.6 <code>load_parent_window</code>	158
12.5.2.7 <code>parent</code>	158
12.5.2.8 <code>save_parent_window</code>	158
12.6 <code>_LibCellRenderer</code> Struct Reference	158
12.6.1 Detailed Description	159

12.6.2 Field Documentation	159
12.6.2.1 super	159
12.7 <code>_RendererSettingsDialog</code> Struct Reference	159
12.7.1 Detailed Description	159
12.7.2 Field Documentation	160
12.7.2.1 <code>cell_height</code>	160
12.7.2.2 <code>cell_width</code>	160
12.7.2.3 <code>layer_check</code>	160
12.7.2.4 <code>parent</code>	160
12.7.2.5 <code>radio_cairo_pdf</code>	160
12.7.2.6 <code>radio_cairo_svg</code>	161
12.7.2.7 <code>radio_latex</code>	161
12.7.2.8 <code>scale</code>	161
12.7.2.9 <code>shape_drawing</code>	161
12.7.2.10 <code>standalone_check</code>	161
12.7.2.11 <code>unit_in_meters</code>	161
12.7.2.12 <code>x_label</code>	162
12.7.2.13 <code>x_output_label</code>	162
12.7.2.14 <code>y_label</code>	162
12.7.2.15 <code>y_output_label</code>	162
12.8 <code>bounding_box::_vectors</code> Struct Reference	162
12.8.1 Detailed Description	163
12.8.2 Field Documentation	163
12.8.2.1 <code>lower_left</code>	163
12.8.2.2 <code>upper_right</code>	163
12.9 <code>application_data</code> Struct Reference	163
12.9.1 Detailed Description	164
12.9.2 Field Documentation	164
12.9.2.1 <code>app</code>	164
12.9.2.2 <code>gui_list</code>	164
12.10 <code>bounding_box</code> Union Reference	164
12.10.1 Detailed Description	165
12.10.2 Field Documentation	165
12.10.2.1 <code>vector_array</code>	165
12.10.2.2 <code>vectors</code>	165
12.11 <code>cairo_layer</code> Struct Reference	165
12.11.1 Detailed Description	166
12.11.2 Field Documentation	166
12.11.2.1 <code>cr</code>	166
12.11.2.2 <code>linfo</code>	166
12.11.2.3 <code>rec</code>	166
12.12 <code>gds_cell</code> Struct Reference	167

12.12.1 Detailed Description	167
12.12.2 Field Documentation	168
12.12.2.1 access_time	168
12.12.2.2 checks	168
12.12.2.3 child_cells	168
12.12.2.4 graphic_objs	168
12.12.2.5 mod_time	168
12.12.2.6 name	169
12.12.2.7 parent_library	169
12.13 gds_cell_checks Struct Reference	169
12.13.1 Detailed Description	170
12.13.2 Field Documentation	170
12.13.2.1 _internal	170
12.13.2.2 affected_by_reference_loop	170
12.13.2.3 unresolved_child_count	170
12.14 gds_cell_instance Struct Reference	171
12.14.1 Detailed Description	171
12.14.2 Field Documentation	172
12.14.2.1 angle	172
12.14.2.2 cell_ref	172
12.14.2.3 flipped	172
12.14.2.4 magnification	172
12.14.2.5 origin	172
12.14.2.6 ref_name	173
12.15 gds_graphics Struct Reference	173
12.15.1 Detailed Description	173
12.15.2 Field Documentation	173
12.15.2.1 datatype	173
12.15.2.2 gfx_type	174
12.15.2.3 layer	174
12.15.2.4 path_render_type	174
12.15.2.5 vertices	174
12.15.2.6 width_absolute	174
12.16 gds_library Struct Reference	175
12.16.1 Detailed Description	175
12.16.2 Field Documentation	175
12.16.2.1 access_time	175
12.16.2.2 cell_names	176
12.16.2.3 cells	176
12.16.2.4 mod_time	176
12.16.2.5 name	176
12.16.2.6 unit_in_meters	176

12.17 gds_point Struct Reference	177
12.17.1 Detailed Description	177
12.17.2 Field Documentation	177
12.17.2.1 x	177
12.17.2.2 y	177
12.18 gds_time_field Struct Reference	177
12.18.1 Detailed Description	178
12.18.2 Field Documentation	178
12.18.2.1 day	178
12.18.2.2 hour	178
12.18.2.3 minute	178
12.18.2.4 month	178
12.18.2.5 second	179
12.18.2.6 year	179
12.19 layer_element_dnd_data Struct Reference	179
12.19.1 Detailed Description	179
12.19.2 Field Documentation	179
12.19.2.1 drag_begin	180
12.19.2.2 drag_data_get	180
12.19.2.3 drag_end	180
12.19.2.4 entries	180
12.19.2.5 entry_count	180
12.20 layer_info Struct Reference	181
12.20.1 Detailed Description	181
12.20.2 Field Documentation	181
12.20.2.1 color	181
12.20.2.2 layer	181
12.20.2.3 name	182
12.20.2.4 stacked_position	182
12.21 render_settings Struct Reference	182
12.21.1 Detailed Description	182
12.21.2 Field Documentation	183
12.21.2.1 renderer	183
12.21.2.2 scale	183
12.21.2.3 tex_pdf_layers	183
12.21.2.4 tex_standalone	183
12.22 tree_stores Struct Reference	184
12.22.1 Detailed Description	184
12.22.2 Field Documentation	184
12.22.2.1 base_store	184
12.22.2.2 base_tree_view	184
12.22.2.3 filter	184

12.22.2.4 search_entry	185
12.23 vector_2d Struct Reference	185
12.23.1 Detailed Description	185
12.23.2 Field Documentation	185
12.23.2.1 x	185
12.23.2.2 y	185
13 File Documentation	187
13.1 bounding-box.c File Reference	187
13.1.1 Detailed Description	188
13.2 bounding-box.c	188
13.3 bounding-box.h File Reference	190
13.3.1 Detailed Description	192
13.4 bounding-box.h	192
13.5 cairo-output.c File Reference	193
13.5.1 Detailed Description	194
13.6 cairo-output.c	194
13.7 cairo-output.h File Reference	197
13.7.1 Detailed Description	198
13.8 cairo-output.h	199
13.9 cairo-renderer.dox File Reference	199
13.10 cell-geometrics.c File Reference	199
13.10.1 Detailed Description	200
13.11 cell-geometrics.c	201
13.12 cell-geometrics.h File Reference	202
13.12.1 Detailed Description	203
13.13 cell-geometrics.h	203
13.14 command-line.c File Reference	204
13.14.1 Detailed Description	204
13.15 command-line.c	205
13.16 command-line.dox File Reference	207
13.17 command-line.h File Reference	207
13.17.1 Detailed Description	208
13.18 command-line.h	208
13.19 compilation.dox File Reference	208
13.20 conv-settings-dialog.c File Reference	208
13.20.1 Detailed Description	210
13.21 conv-settings-dialog.c	210
13.22 conv-settings-dialog.h File Reference	215
13.22.1 Detailed Description	216
13.23 conv-settings-dialog.h	216
13.24 external-renderer.c File Reference	217

13.24.1 Detailed Description	218
13.25 external-renderer.c	218
13.26 external-renderer.dox File Reference	219
13.27 external-renderer.h File Reference	219
13.27.1 Detailed Description	221
13.28 external-renderer.h	221
13.29 gds-parser.c File Reference	221
13.29.1 Detailed Description	223
13.30 gds-parser.c	224
13.31 gds-parser.h File Reference	233
13.31.1 Detailed Description	234
13.32 gds-parser.h	235
13.33 gds-render-gui.c File Reference	235
13.33.1 Detailed Description	236
13.34 gds-render-gui.c	237
13.35 gds-render-gui.h File Reference	243
13.35.1 Detailed Description	244
13.36 gds-render-gui.h	244
13.37 gds-tree-checker.c File Reference	244
13.37.1 Detailed Description	245
13.38 gds-tree-checker.c	246
13.39 gds-tree-checker.h File Reference	248
13.39.1 Detailed Description	248
13.40 gds-tree-checker.h	249
13.41 gds-types.h File Reference	249
13.41.1 Detailed Description	250
13.42 gds-types.h	251
13.43 geometric.dox File Reference	252
13.44 gpl-2.0.md File Reference	252
13.45 gui.dox File Reference	252
13.46 latex-output.c File Reference	252
13.46.1 Detailed Description	253
13.47 latex-output.c	253
13.48 latex-output.h File Reference	256
13.48.1 Detailed Description	257
13.49 latex-output.h	257
13.50 latex-renderer.dox File Reference	258
13.51 layer-element.c File Reference	258
13.51.1 Detailed Description	259
13.52 layer-element.c	259
13.53 layer-element.h File Reference	261
13.53.1 Detailed Description	262

13.54 layer-element.h	263
13.55 layer-info.c File Reference	264
13.55.1 Detailed Description	264
13.55.2 Function Documentation	264
13.55.2.1 layer_info_delete_struct()	264
13.56 layer-info.c	265
13.57 layer-info.h File Reference	265
13.57.1 Detailed Description	266
13.57.2 Function Documentation	267
13.57.2.1 layer_info_delete_struct()	267
13.58 layer-info.h	267
13.59 layer-selector.c File Reference	268
13.59.1 Detailed Description	269
13.60 layer-selector.c	270
13.61 layer-selector.dox File Reference	278
13.62 layer-selector.h File Reference	278
13.62.1 Detailed Description	279
13.63 layer-selector.h	280
13.64 lib-cell-renderer.c File Reference	280
13.64.1 Detailed Description	281
13.65 lib-cell-renderer.c	282
13.66 lib-cell-renderer.dox File Reference	283
13.67 lib-cell-renderer.h File Reference	283
13.67.1 Detailed Description	285
13.68 lib-cell-renderer.h	285
13.69 lmf-spec.dox File Reference	285
13.70 main-page.dox File Reference	285
13.71 main.c File Reference	285
13.71.1 Detailed Description	287
13.71.2 Function Documentation	287
13.71.2.1 app_about()	287
13.71.2.2 app_quit()	287
13.71.2.3 gapp_activate()	288
13.71.2.4 gui_window_closed_callback()	289
13.71.2.5 main()	289
13.71.2.6 print_version()	290
13.71.2.7 start_gui()	291
13.71.3 Variable Documentation	292
13.71.3.1 app_actions	292
13.72 main.c	292
13.73 mapping-parser.c File Reference	295
13.73.1 Detailed Description	296

13.74 mapping-parser.c	297
13.75 mapping-parser.h File Reference	298
13.75.1 Detailed Description	299
13.76 mapping-parser.h	300
13.77 README.MD File Reference	300
13.78 README.MD	300
13.79 renderers.dox File Reference	300
13.80 tree-store.c File Reference	300
13.81 tree-store.c	301
13.82 tree-store.h File Reference	303
13.82.1 Detailed Description	304
13.83 tree-store.h	304
13.84 usage.dox File Reference	305
13.85 vector-operations.c File Reference	305
13.85.1 Detailed Description	306
13.86 vector-operations.c	306
13.87 vector-operations.h File Reference	308
13.87.1 Detailed Description	309
13.88 vector-operations.h	309
13.89 version.c File Reference	310
13.90 version.c	310
13.91 version.h File Reference	310
13.92 version.h	311
13.93 versioning.dox File Reference	311
13.94 widgets.dox File Reference	311

Chapter 1

Main Page

This program converts GDS layout files to

- PDF Files using the [Cairo Renderer](#)
- Latex code (TikZ) using the [LaTeX/TikZ Renderer](#)

See the [Usage](#) page for details and [Compilation](#) for building instructions and [Version Number](#) for the versioning scheme of this program.

Chapter 2

Compilation

2.1 Preface

GDS-Render is designed for UNIX-like, especially GNU/Linux based systems. It was developed under a Linux system. Therefore, best performance is expected using a Linux operating system.

2.2 Dependencies

The dependencies of GDS-Render are:

2.2.1 Program Dependencies

- GLib2
- GTK3
- Cairographics

2.2.2 Compilation Dependencies

These dependencies are not needed for running the program; just for compilation.

- Build System (GCC + binutils, make, etc...). Most distributions supply a "development" meta-package containing this stuff.
- cmake \geq 2.8
- More or less optional: git. Used for extraction of the precise version number. It is strongly recommended to provide git!
- Optional: doxygen for this nice documentation.

The dependency list of GTK3 already includes Cairographics and GLib2. You should be on the safe side with a recent GTK3 version.

Development is done with the following library versions:

Cairographics	Glib2	GTK3
1.16.0-2	2.60.0-1	3.↔ 24.7

2.3 Compilation Instructions

2.3.1 General Linux Build Instruction

Go to the build directory you want to compile in. This may be the gds-render project root. Execute
`cmake <Path to gds-render root>`

Cmake will check the dependencies. Once cmake has finished. Type
`make`

in order to build the program and
`make documentation`

to build the doxygen documentation.

2.3.2 Archlinux Package

The subfolder 'AUR' contains a PKGBUILD file to build an Archlinux/Pacman package.

2.3.3 Warnings

The compiler will throw the following warnings. Compiled with GCC 8.2.1.

Warning	Assessment
warning: 'calculate_path_miter_points' defined but not used [-Wunused-function]	Ignore. Function will be used in later versions.

Chapter 3

Layer Mapping File Specification

File Format

The layer mapping file contains information on how to render the layers. The information is stored in CSV format – *True CSV*; not that rubbish with semicolons that Excel calls CSV.

Each line representing a layer consists of following fields:

layer,r,g,b,a,export,name

- **layer**: Layer number identifying this layer.
- **r,b,g,a**: RGBA color value using double precision float values in the range from 0 to 1.
- **export**: Either '1' or '0'. Defining whether to render this layer into the output file.
- **name**: The name of the layer.

the order of the layers inside the layer mapping file defines the layer stack in the rendered output. The first layer is at the bottom, the last at the top.

Handling Inside the GUI

The layer mapping file can be imported and exported inside the GUI.

Export

During export, all layer configurations are written to the mapping file

Import

During import, all layer configurations are loaded from the mapping file. This overwrites any configuration done to that layer. Layers that are not present in the layer mapping file are appended at the end of the list. This means, they are rendered on top of the other layers. Because the layer mapping file does not contain any information on these layers, their configuration is not reset during import.

Chapter 4

Usage

4.1 Command Line Interface

To use the application on the command line check 'gds-render --help'.

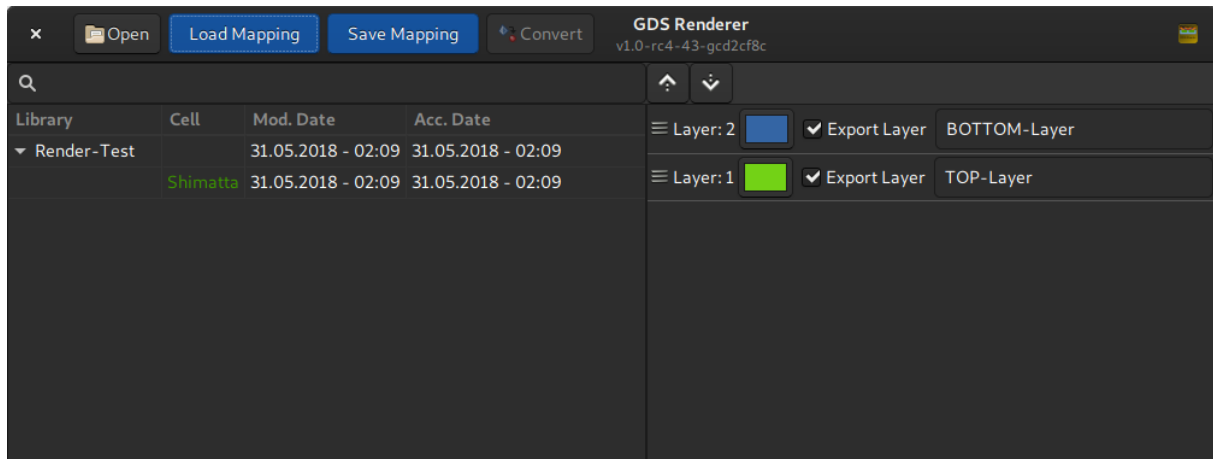
Application Options:

- -t, --tikz Output TikZ code
- -p, --pdf Output PDF document
- -s, --scale=SCALE Divide output coordinates by SCALE
- -o, --tex-output=PATH Optional path for TeX file
- -O, --pdf-output=PATH Optional path for PDF file
- -m, --mapping=PATH Path for Layer Mapping File
- -c, --cell=NAME Cell to render
- -a, --tex-standalone Create standalone PDF
- -l, --tex-layers Create PDF Layers (OCG)
- -P, --custom-render-lib=PATH Path to a custom shared object, that implements the `render_cell_to_file` function
- -e, --external-lib-output=PATH Output path for external render library
- --display=DISPLAY X display to use

4.2 Graphical User Interface

The graphical user interface (GUI) can be used to open GDS Files, configure the layer rendering (colors, order, transparency etc.), and convert cells.

It is possible to export the layer configurations so they can be used later on. Even in the [Command Line Interface](#)



The cell selector on the left shows the GDS Libraries and Cells. The cells are marked green if all references inside the cell could be found. If not all references could be found, the cell is marked orange. This doesn't show if child cells have missing childs. Only one level of the hierarchy is checked in order to make it easier to spot an erroneous cell. Cells with missing child cells are still renderable but – obviously – faulty. If a cell or any sub-cell contains a reference loop, the cell is marked red. In this case it can't be selected for rendering.

In the above image the cell is green; so everything is okay.

Chapter 5

Version Number

5.1 Main Versioning Scheme

The version number of this application consists of a given version in the format of 'v1.0'. Where the first number indicates a major release and the second number indicates minor changes.

Versions, including release candidates and path-levels, are tagged in git.

5.1.1 Release Candidates

Release candidates are software versions that seem stable and functional to become a new version but testing is not fully finished. These versions are marked with an '-rcX', where X is the number of the release candidate. The 3rd release candidate of version 4.2 would be 'v4.2-rc3'. Release candidates are in a frozen state. Only bugfixes that are necessary for functionality are applied to these versions before releasing the final version.

5.1.2 Patch Levels

If an already released version contains bugs that need to be fixed, the version number is not incremented. Instead a new version number with a patch-level is created. The patch-level is appended with a dash directly after the version number. The first patch-level of version 3.5 would be: 'v3.5-1'.

5.2 Git Based Version Number

The application and this documentation contain a git-based version number. With this version number not only released versions but all development points of the software can be uniquely identified.

An example for such a version number is: *v1.0-rc4-41-gaa41373-dirty*

It consists of the last [Main Versioning Scheme](#) (in this case version 1.0 – Release candidate 4) and some other information from the source code management system. The number after the version tag is the commit count after the given version. In this case the specified version is 41 commits after the last tagged version 'v1.0-rc4'. The next section always starts with a 'g' (for git) and after that contains the first letters of the commit ID. In this case an additional '-dirty' is appended, showing that the software version contains unstaged changes.

In tabular form: *v1.0-rc4-41-gaa41373-dirty*

Last tagged version	Commits since that version	Start of commit ID	Unstaged changes?
1.0-rc4	41	aa41373	yes

This git-based version number is automatically put into the application and this doxumentation during the application's compilation / the documentation's generation. For this *git* is needed. Therefore, it is highly recommended to have 'git' installed for compilation although it is no build dependency. In case of a missing git installation, the string "! version not set !" is compiled into the application.

Chapter 6

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

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

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

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) yyyy name of author
```

```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.
```

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.
```

```
signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the [GNU Lesser General Public License](#) instead of this License.

Chapter 7

GDS-Render Readme

This software is a rendering program for GDS2 layout files. The GDS2 format is mainly used in integrated circuit development. This program allows the conversion of a GDS file to a vector graphics file.

Output Formats

- Export GDS Layout to LaTeX (using TikZ).
- Export to PDF (Cairographics).

Features

Note: Due to various size limitations of both TikZ and the PDF export, the layout might not render correctly. In this case adjust the scale value. A higher scale value scales down your design.

- Configurable layer stack-up.
- Layer colors configurable as ARGB color values.
- Command line interface.
- **Awesome Somehow usable GUI.**

License and Other Stuff

- Free software (GPLv2 *only*)
- Coded in plain C using GTK+3.0, Glib2, and Cairographics

Chapter 8

Module Index

8.1 Modules

Here is a list of all modules:

Command Line Interface	30
Geometric Helper Functions	35
Graphical User Interface	49
LayerSelector Object	71
LibCellRenderer GObject	93
Custom GTK Widgets	100
RendererSettingsDialog	130
LayerElement	142
Output Renderers	99
Cairo Renderer	25
External Shared Object Renderer	33
LaTeX/TikZ Renderer	64
GDS-Utilities	101
Mapping-Parser	126
Version Number	129

Chapter 9

Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

gds_cell_checks::_check_internals	
For the internal use of the checker	151
_GdsRenderGui	152
_LayerElement	154
_LayerElementPriv	155
_LayerSelector	157
_LibCellRenderer	158
_RendererSettingsDialog	159
bounding_box::_vectors	162
application_data	
Structure containing The GtkApplication and a list containing the GdsRenderGui objects	163
bounding_box	164
cairo_layer	
The cairo_layer struct Each rendered layer is represented by this struct	165
gds_cell	
A Cell inside a gds_library	167
gds_cell_checks	
Stores the result of the cell checks	169
gds_cell_instance	
This represents an instanc of a cell inside another cell	171
gds_graphics	
A GDS graphics object	173
gds_library	
GDS Toplevel library	175
gds_point	
A point in the 2D plane. Sometimes references as vertex	177
gds_time_field	
Date information for cells and libraries	177
layer_element_dnd_data	
This structure holds the necessary data to set up a LayerElement for Drag'n'Drop	179
layer_info	
Layer information	181
render_settings	
This struct holds the renderer configuration	182
tree_stores	184
vector_2d	185

Chapter 10

File Index

10.1 File List

Here is a list of all files with brief descriptions:

bounding-box.c	Calculation of bounding boxes	187
bounding-box.h	Header for calculation of bounding boxes	190
cairo-output.c	Output renderer for Cairo PDF export	193
cairo-output.h	Header File for Cairo output renderer	197
cell-geometrics.c	Calculation of gds_cell trigonometrics	199
cell-geometrics.h	Calculation of gds_cell geometrics	202
command-line.c	Function to render according to command line parameters	204
command-line.h	Render according to command line parameters	207
conv-settings-dialog.c	Implementation of the setting dialog	208
conv-settings-dialog.h	Header file for the Conversion Settings Dialog	215
external-renderer.c	This file implements the dynamic library loading for the external rendering feature	217
external-renderer.h	Render according to command line parameters	219
gds-parser.c	Implementation of the GDS-Parser	221
gds-parser.h	Header file for the GDS-Parser	233
gds-render-gui.c	Handling of GUI	235
gds-render-gui.h	Header for GdsRenderGui Object	243
gds-tree-checker.c	Checking functions of a cell tree	244
gds-tree-checker.h	Checking functions of a cell tree (Header)	248

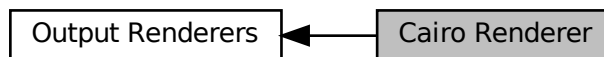
gds-types.h	Defines types and macros used by the GDS-Parser	249
latex-output.c	LaTeX output renderer	252
latex-output.h	LaTeX output renderer	256
layer-element.c	Implementation of the layer element used for configuring layer colors etc	258
layer-element.h	Implementation of the layer element used for configuring layer colors etc	261
layer-info.c	Helper functions for layer info struct	264
layer-info.h	Helper functions and definition of layer info struct	265
layer-selector.c	Implementation of the layer selector	268
layer-selector.h	Implementation of the Layer selection list	278
lib-cell-renderer.c	LibCellRenderer GObject Class	280
lib-cell-renderer.h	Header file for the LibCellRenderer GObject Class	283
main.c	Main.c	285
mapping-parser.c	Function to read a mapping file line and parse it	295
mapping-parser.h	Function to read a mapping file line and parse it	298
README.MD	300
tree-store.c	300
tree-store.h	Header file for Tree store implementation	303
vector-operations.c	2D Vector operations	305
vector-operations.h	Header for 2D Vector operations	308
version.c	310
version.h	310

Chapter 11

Module Documentation

11.1 Cairo Renderer

Collaboration diagram for Cairo Renderer:



Data Structures

- struct [cairo_layer](#)

The [cairo_layer](#) struct Each rendered layer is represented by this struct.

Macros

- #define [MAX_LAYERS](#) (300)

Maximum layer count the output renderer can process. Typically GDS only specifies up to 255 layers.

Functions

- static void [revert_inherited_transform](#) (struct [cairo_layer](#) *layers)
Revert the last transformation on all layers.
- static void [apply_inherited_transform_to_all_layers](#) (struct [cairo_layer](#) *layers, const struct [gds_point](#) *origin, double magnification, gboolean flipping, double rotation, double scale)
Applies transformation to all layers.
- static void [render_cell](#) (struct [gds_cell](#) *cell, struct [cairo_layer](#) *layers, double scale)
render_cell Render a cell with its sub-cells
- void [cairo_render_cell_to_vector_file](#) (struct [gds_cell](#) *cell, GList *layer_infos, char *pdf_file, char *svg_file, double scale)
Render cell to a PDF file specified by pdf_file.

11.1.1 Detailed Description

11.1.2 Macro Definition Documentation

11.1.2.1 MAX_LAYERS

```
#define MAX_LAYERS (300)
```

Maximum layer count the output renderer can process. Typically GDS only specifies up to 255 layers.

Definition at line 34 of file [cairo-output.h](#).

11.1.3 Function Documentation

11.1.3.1 apply_inherited_transform_to_all_layers()

```
static void apply_inherited_transform_to_all_layers (
    struct cairo\_layer * layers,
    const struct gds\_point * origin,
    double magnification,
    gboolean flipping,
    double rotation,
    double scale ) [static]
```

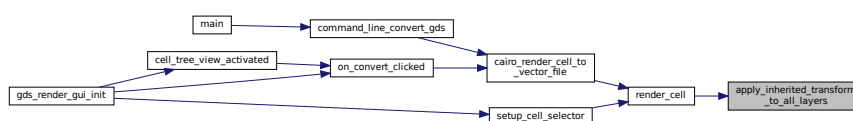
Applies transformation to all layers.

Parameters

<i>layers</i>	Array of layers
<i>origin</i>	Origin translation
<i>magnification</i>	Scaling
<i>flipping</i>	Mirror image on x-axis before rotating
<i>rotation</i>	Rotattion in degrees
<i>scale</i>	Scale the image down by. Only used for sclaing origin coordinates. Not applied to layer.

Definition at line 71 of file [cairo-output.c](#).

Here is the caller graph for this function:



11.1.3.2 cairo_render_cell_to_vector_file()

```
void cairo_render_cell_to_vector_file (
    struct gds_cell * cell,
    GList * layer_infos,
    char * pdf_file,
    char * svg_file,
    double scale )
```

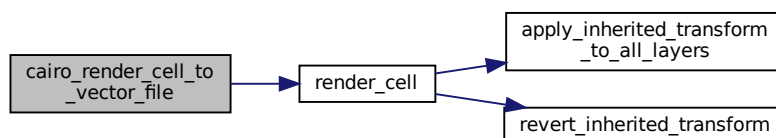
Render `cell` to a PDF file specified by `pdf_file`.

Parameters

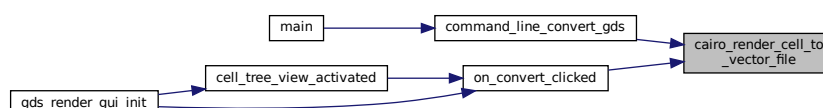
<code>cell</code>	Toplevel cell to Cairo Renderer
<code>layer_infos</code>	List of layer information. Specifies color and layer stacking
<code>pdf_file</code>	PDF output file. Set to NULL if no PDF file has to be generated
<code>svg_file</code>	SVG output file. Set to NULL if no SVG file has to be generated
<code>scale</code>	Scale the output image down by <code>scale</code>

Definition at line [182](#) of file [cairo-output.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.1.3.3 render_cell()

```
static void render_cell (
    struct gds_cell * cell,
    struct cairo_layer * layers,
    double scale ) [static]
```

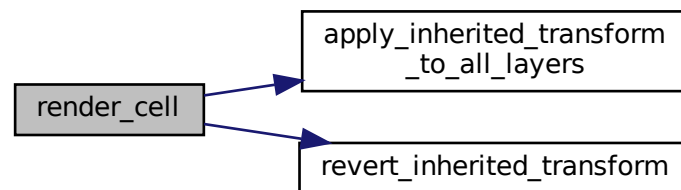
render_cell Render a cell with its sub-cells

Parameters

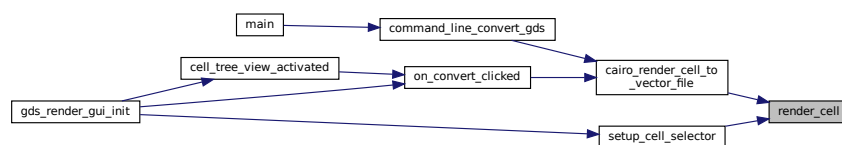
<i>cell</i>	Cell to render
<i>layers</i>	Cell will be rendered into these layers
<i>scale</i>	scale image down by this factor

Definition at line 101 of file [cairo-output.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.1.3.4 revert_inherited_transform()

```
static void revert_inherited_transform (
    struct cairo_layer * layers ) [static]
```

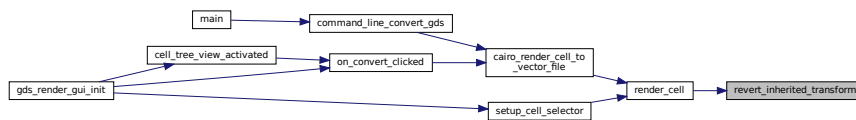
Revert the last transformation on all layers.

Parameters

<i>layers</i>	Pointer to cairo_layer structures
---------------	---

Definition at line 51 of file [cairo-output.c](#).

Here is the caller graph for this function:



11.2 Command Line Interface

Functions

- static void `delete_layer_info_with_name` (struct `layer_info` *info)
Delete `layer_info` and free nem element.
- void `command_line_convert_gds` (char *gds_name, char *pdf_name, char *tex_name, gboolean pdf, gboolean tex, char *layer_file, char *cell_name, double scale, gboolean pdf_layers, gboolean pdf_↔standalone, gboolean svg, char *svg_name, char *so_name, char *so_out_file)
Convert GDS according to supplied parameters.

11.2.1 Detailed Description

11.2.2 Function Documentation

11.2.2.1 `command_line_convert_gds()`

```
void command_line_convert_gds (
    char * gds_name,
    char * pdf_name,
    char * tex_name,
    gboolean pdf,
    gboolean tex,
    char * layer_file,
    char * cell_name,
    double scale,
    gboolean pdf_layers,
    gboolean pdf_standalone,
    gboolean svg,
    char * svg_name,
    char * so_name,
    char * so_out_file )
```

Convert GDS according to supplied parameters.

Parameters

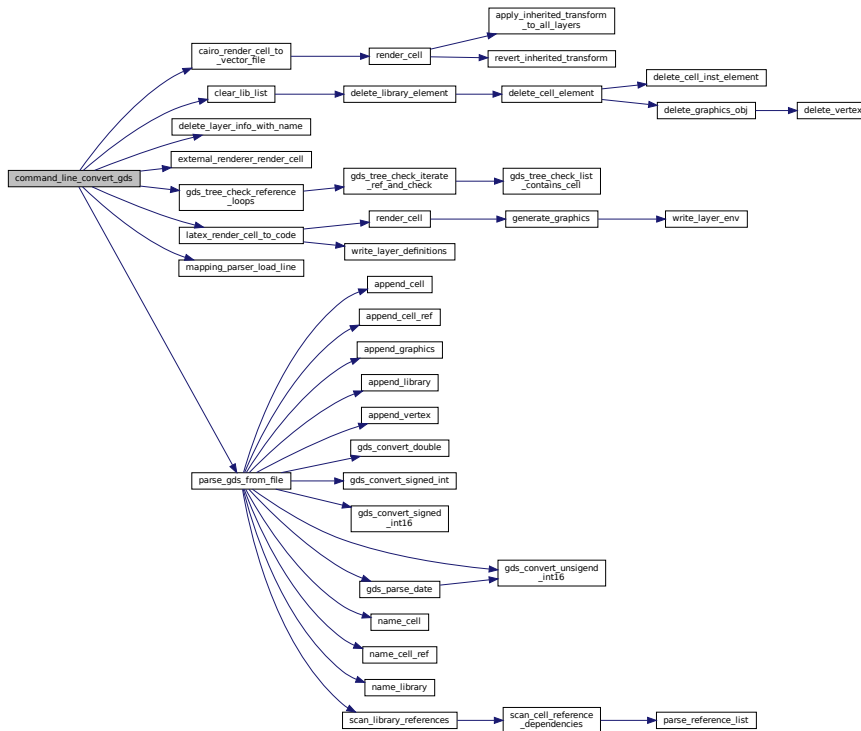
<code>gds_name</code>	GDS File path
<code>pdf_name</code>	Cairo-PDF path
<code>tex_name</code>	TeX/TikZ path
<code>pdf</code>	Render Cairo
<code>tex</code>	Render LaTeX
<code>layer_file</code>	Layer mapping file
<code>cell_name</code>	Cell name to render
<code>scale</code>	Scale image down by this value
<code>pdf_layers</code>	TikZ creates OCG layers
<code>pdf_standalone</code>	LaTeX document is standalone?
<code>svg</code>	Render to SVG file
<code>so_name</code>	Path to shared object of custom renderer
<code>so_out_file</code>	Output file path for custom renderer
<code>svg_name</code>	SVG file name

Note

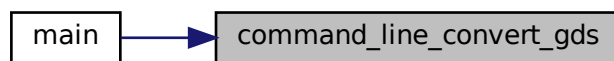
This function is pretty damn retarded (Lots of parameters). Will be reworked when generating GObjects for renderers.

Definition at line 58 of file [command-line.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.2.2.2 delete_layer_info_with_name()

```

static void delete_layer_info_with_name (
    struct layer_info * info ) [static]
    
```

Delete [layer_info](#) and free nem element.

Like [delete_layer_info_struct\(\)](#) but also frees [layer_info::name](#)

Parameters

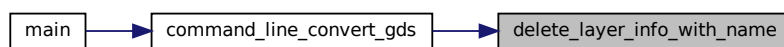
<i>info</i>	
-------------	--

Warning

This function must not be used if the `layer_info::name` field references the internal storage strings if e.g. an entry field

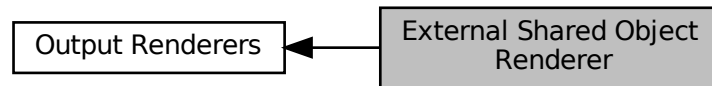
Definition at line 49 of file `command-line.c`.

Here is the caller graph for this function:



11.3 External Shared Object Renderer

Collaboration diagram for External Shared Object Renderer:



Macros

- `#define EXTERNAL_LIBRARY_FUNCTION "render_cell_to_file"`
function name expected to be found in external library.

Functions

- `int external_renderer_render_cell (struct gds_cell *toplevel_cell, GList *layer_info_list, char *output_file, char *so_path)`
external_renderer_render_cell

11.3.1 Detailed Description

11.3.2 Macro Definition Documentation

11.3.2.1 EXTERNAL_LIBRARY_FUNCTION

```
#define EXTERNAL_LIBRARY_FUNCTION "render_cell_to_file"
```

function name expected to be found in external library.

The function has to be defined as follows:

```
int function_name(gds_cell *toplevel, GList *layer_info_list, char *output_file_name)
```

Definition at line 45 of file [external-renderer.h](#).

11.3.3 Function Documentation

11.3.3.1 external_renderer_render_cell()

```
int external_renderer_render_cell (
    struct gds_cell * toplevel_cell,
    GList * layer_info_list,
    char * output_file,
    char * so_path )
```

`external_renderer_render_cell`

Parameters

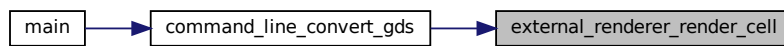
<i>toplevel_cell</i>	The toplevel cell to render
<i>layer_info_list</i>	The layer information. Contains layer_info elements
<i>output_file</i>	Output file
<i>so_path</i>	Path to the shared object file containing EXTERNAL_LIBRARY_FUNCTION

Returns

0 on success

Definition at line 36 of file [external-renderer.c](#).

Here is the caller graph for this function:



11.4 Geometric Helper Functions

The geometric helper function are used to calculate bounding boxes.

Data Structures

- union [bounding_box](#)
- struct [vector_2d](#)

Macros

- #define [MIN](#)(a, b) (((a) < (b)) ? (a) : (b))
Return smaller number.
- #define [MAX](#)(a, b) (((a) > (b)) ? (a) : (b))
Return bigger number.
- #define [ABS_DBL](#)(a) ((a) < 0 ? -(a) : (a))
- #define [ABS_DBL](#)(a) ((a) < 0 ? -(a) : (a))
- #define [DEG2RAD](#)(a) ((a)*M_PI/180.0)

Typedefs

- typedef void(* [conv_generic_to_vector_2d_t](#)) (void *, struct [vector_2d](#) *)

Functions

- void [bounding_box_calculate_polygon](#) (GList *vertices, [conv_generic_to_vector_2d_t](#) conv_func, union [bounding_box](#) *box)
- void [bounding_box_update_box](#) (union [bounding_box](#) *destination, union [bounding_box](#) *update)
- void [bounding_box_prepare_empty](#) (union [bounding_box](#) *box)
- static void [calculate_path_miter_points](#) (struct [vector_2d](#) *a, struct [vector_2d](#) *b, struct [vector_2d](#) *c, struct [vector_2d](#) *m1, struct [vector_2d](#) *m2, double width)
- void [bounding_box_calculate_path_box](#) (GList *vertices, double thickness, [conv_generic_to_vector_2d_t](#) conv_func, union [bounding_box](#) *box)
- void [bounding_box_update_point](#) (union [bounding_box](#) *destination, [conv_generic_to_vector_2d_t](#) conv_func, void *pt)
- void [bounding_box_apply_transform](#) (double scale, double rotation_deg, bool flip_at_x, union [bounding_box](#) *box)
Apply transformations onto bounding box.
- static void [convert_gds_point_to_2d_vector](#) (struct [gds_point](#) *pt, struct [vector_2d](#) *vector)
- static void [update_box_with_gfx](#) (union [bounding_box](#) *box, struct [gds_graphics](#) *gfx)
Update the given bounding box with the bounding box of a graphics element.
- void [calculate_cell_bounding_box](#) (union [bounding_box](#) *box, struct [gds_cell](#) *cell)
calculate_cell_bounding_box Calculate bounding box of gds cell
- double [vector_2d_scalar_multiply](#) (struct [vector_2d](#) *a, struct [vector_2d](#) *b)
- void [vector_2d_normalize](#) (struct [vector_2d](#) *vec)
- void [vector_2d_rotate](#) (struct [vector_2d](#) *vec, double angle)
- struct [vector_2d](#) * [vector_2d_copy](#) (struct [vector_2d](#) *opt_res, struct [vector_2d](#) *vec)
- struct [vector_2d](#) * [vector_2d_alloc](#) (void)
- void [vector_2d_free](#) (struct [vector_2d](#) *vec)
- void [vector_2d_scale](#) (struct [vector_2d](#) *vec, double scale)
- double [vector_2d_abs](#) (struct [vector_2d](#) *vec)
- double [vector_2d_calculate_angle_between](#) (struct [vector_2d](#) *a, struct [vector_2d](#) *b)
- void [vector_2d_subtract](#) (struct [vector_2d](#) *res, struct [vector_2d](#) *a, struct [vector_2d](#) *b)
- void [vector_2d_add](#) (struct [vector_2d](#) *res, struct [vector_2d](#) *a, struct [vector_2d](#) *b)

11.4.1 Detailed Description

The geometric helper function are used to calculate bounding boxes.

Warning

Code is incomplete. Please double check for functionality!

11.4.2 Macro Definition Documentation

11.4.2.1 ABS_DBL [1/2]

```
#define ABS_DBL(  
    a ) ((a) < 0 ? -(a) : (a))
```

Definition at line 36 of file [vector-operations.c](#).

11.4.2.2 ABS_DBL [2/2]

```
#define ABS_DBL(  
    a ) ((a) < 0 ? -(a) : (a))
```

Definition at line 38 of file [bounding-box.c](#).

11.4.2.3 DEG2RAD

```
#define DEG2RAD(  
    a ) ((a)*M_PI/180.0)
```

Definition at line 42 of file [vector-operations.h](#).

11.4.2.4 MAX

```
#define MAX(  
    a,  
    b ) (((a) > (b)) ? (a) : (b))
```

Return bigger number.

Definition at line 37 of file [bounding-box.c](#).

11.4.2.5 MIN

```
#define MIN(
    a,
    b ) ((a) < (b)) ? (a) : (b))
```

Return smaller number.

Definition at line 36 of file [bounding-box.c](#).

11.4.3 Typedef Documentation

11.4.3.1 conv_generic_to_vector_2d_t

```
typedef void(* conv_generic_to_vector_2d_t) (void *, struct vector_2d *)
```

Definition at line 47 of file [bounding-box.h](#).

11.4.4 Function Documentation

11.4.4.1 bounding_box_apply_transform()

```
void bounding_box_apply_transform (
    double scale,
    double rotation_deg,
    bool flip_at_x,
    union bounding_box * box )
```

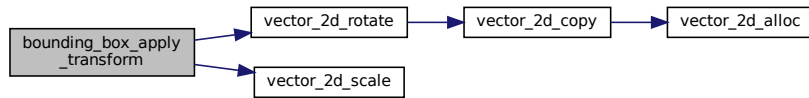
Apply transformations onto bounding box.

Parameters

<i>scale</i>	Scaling factor
<i>rotation_deg</i>	Roation of bounding box around the origin in degrees (counterclockwise)
<i>flip_at_x</i>	Flip the boundig box on the x axis before rotating.
<i>box</i>	Bounding box the operations should be applied to.

Definition at line 190 of file [bounding-box.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



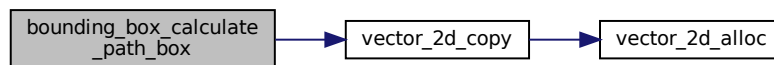
11.4.4.2 bounding_box_calculate_path_box()

```

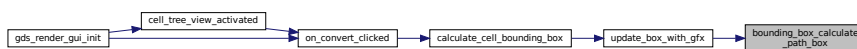
void bounding_box_calculate_path_box (
    GList * vertices,
    double thickness,
    conv_generic_to_vector_2d_t conv_func,
    union bounding_box * box )
  
```

Definition at line 137 of file [bounding-box.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

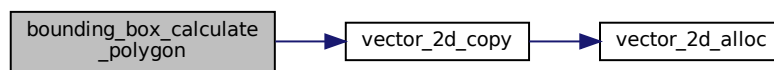


11.4.4.3 bounding_box_calculate_polygon()

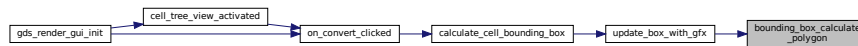
```
void bounding_box_calculate_polygon (
    GList * vertices,
    conv_generic_to_vector_2d_t conv_func,
    union bounding_box * box )
```

Definition at line 40 of file [bounding-box.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.4.4.4 bounding_box_prepare_empty()

```
void bounding_box_prepare_empty (
    union bounding_box * box )
```

Definition at line 86 of file [bounding-box.c](#).

Here is the caller graph for this function:

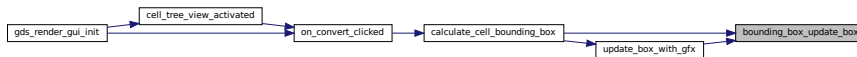


11.4.4.5 bounding_box_update_box()

```
void bounding_box_update_box (
    union bounding_box * destination,
    union bounding_box * update )
```

Definition at line 71 of file [bounding-box.c](#).

Here is the caller graph for this function:

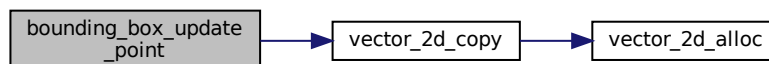


11.4.4.6 bounding_box_update_point()

```
void bounding_box_update_point (
    union bounding_box * destination,
    conv_generic_to_vector_2d_t conv_func,
    void * pt )
```

Definition at line 165 of file [bounding-box.c](#).

Here is the call graph for this function:



11.4.4.7 calculate_cell_bounding_box()

```
void calculate_cell_bounding_box (
    union bounding_box * box,
    struct gds_cell * cell )
```

`calculate_cell_bounding_box` Calculate bounding box of gds cell

Parameters

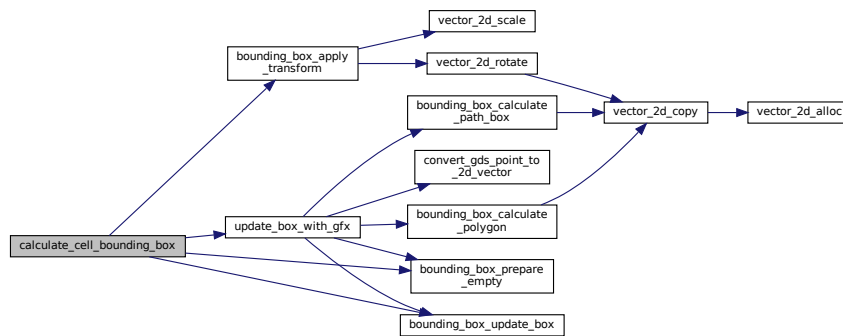
<i>box</i>	Resulting boundig box. Will be udated and not overwritten
<i>cell</i>	Toplevel cell

Warning

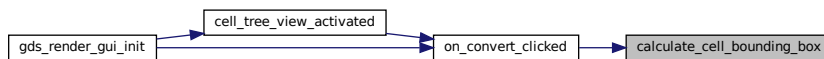
Path handling not yet implemented correctly.

Definition at line 80 of file [cell-geometrics.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

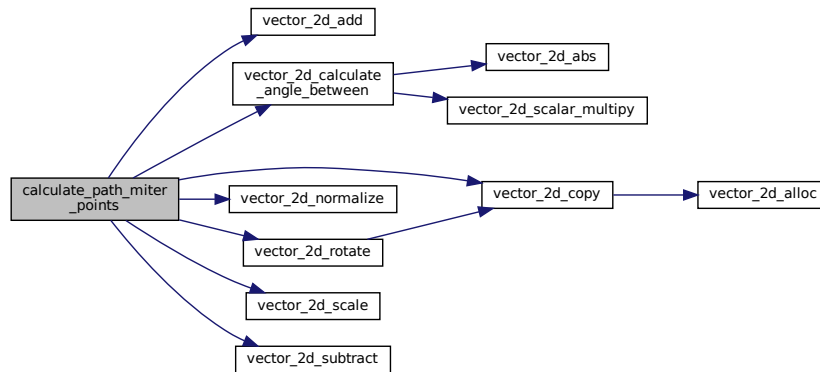
**11.4.4.8 calculate_path_miter_points()**

```

static void calculate_path_miter_points (
    struct vector_2d * a,
    struct vector_2d * b,
    struct vector_2d * c,
    struct vector_2d * m1,
    struct vector_2d * m2,
    double width ) [static]
  
```

Definition at line 94 of file [bounding-box.c](#).

Here is the call graph for this function:



11.4.4.9 convert_gds_point_to_2d_vector()

```

static void convert_gds_point_to_2d_vector (
    struct gds_point * pt,
    struct vector_2d * vector ) [static]
  
```

Definition at line 35 of file [cell-geometrics.c](#).

Here is the caller graph for this function:



11.4.4.10 update_box_with_gfx()

```

static void update_box_with_gfx (
    union bounding_box * box,
    struct gds_graphics * gfx ) [static]
  
```

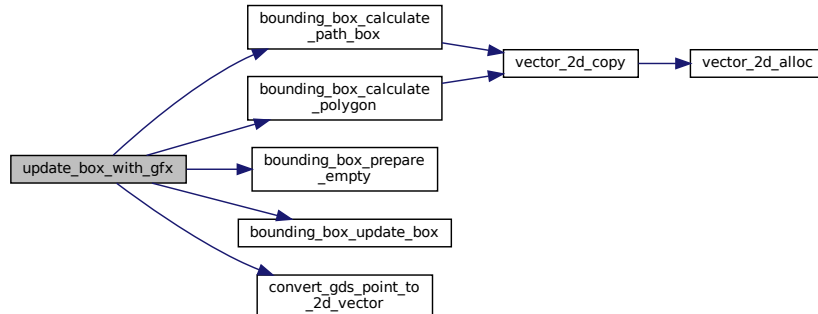
Update the given bounding box with the bounding box of a graphics element.

Parameters

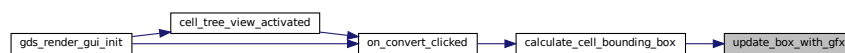
<i>box</i>	box to update
<i>gfx</i>	Graphics element

Definition at line 46 of file [cell-geometrics.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

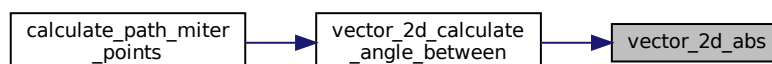


11.4.4.11 vector_2d_abs()

```
double vector_2d_abs (
    struct vector_2d * vec )
```

Definition at line 114 of file [vector-operations.c](#).

Here is the caller graph for this function:

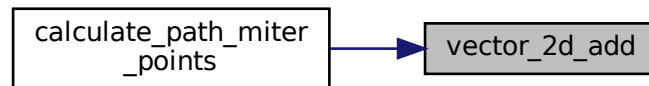


11.4.4.12 vector_2d_add()

```
void vector_2d_add (
    struct vector_2d * res,
    struct vector_2d * a,
    struct vector_2d * b )
```

Definition at line 142 of file [vector-operations.c](#).

Here is the caller graph for this function:

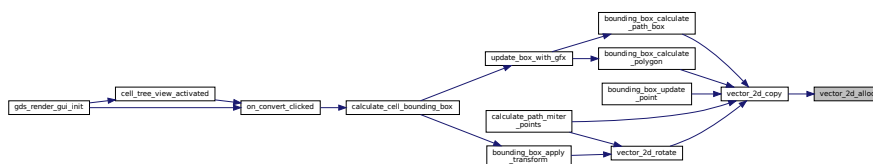


11.4.4.13 vector_2d_alloc()

```
struct vector_2d * vector_2d_alloc (
    void )
```

Definition at line 93 of file [vector-operations.c](#).

Here is the caller graph for this function:

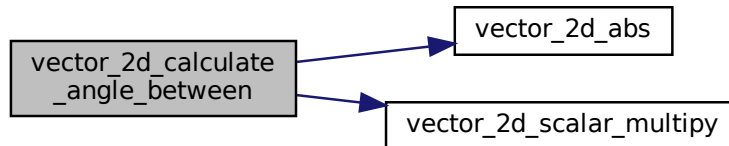


11.4.4.14 `vector_2d_calculate_angle_between()`

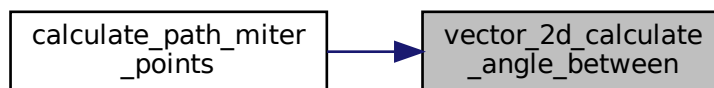
```
double vector_2d_calculate_angle_between (
    struct vector_2d * a,
    struct vector_2d * b )
```

Definition at line 123 of file [vector-operations.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.4.4.15 `vector_2d_copy()`

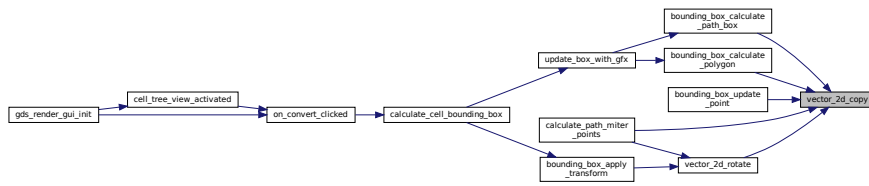
```
struct vector_2d * vector_2d_copy (
    struct vector_2d * opt_res,
    struct vector_2d * vec )
```

Definition at line 74 of file [vector-operations.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.4.4.16 vector_2d_free()

```
void vector_2d_free (
    struct vector_2d * vec )
```

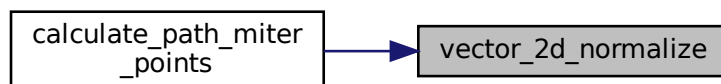
Definition at line 98 of file [vector-operations.c](#).

11.4.4.17 vector_2d_normalize()

```
void vector_2d_normalize (
    struct vector_2d * vec )
```

Definition at line 46 of file [vector-operations.c](#).

Here is the caller graph for this function:

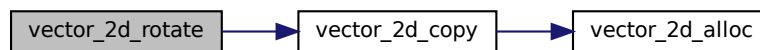


11.4.4.18 vector_2d_rotate()

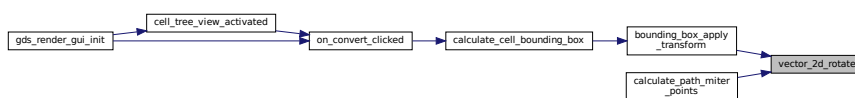
```
void vector_2d_rotate (
    struct vector_2d * vec,
    double angle )
```

Definition at line 56 of file [vector-operations.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

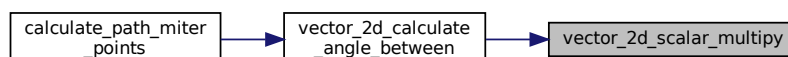


11.4.4.19 vector_2d_scalar_multiply()

```
double vector_2d_scalar_multiply (
    struct vector_2d * a,
    struct vector_2d * b )
```

Definition at line 38 of file [vector-operations.c](#).

Here is the caller graph for this function:

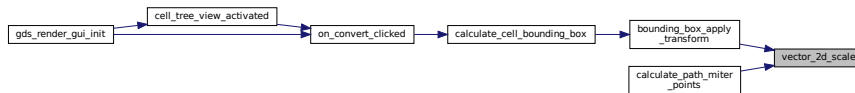


11.4.4.20 `vector_2d_scale()`

```
void vector_2d_scale (
    struct vector_2d * vec,
    double scale )
```

Definition at line 105 of file `vector-operations.c`.

Here is the caller graph for this function:

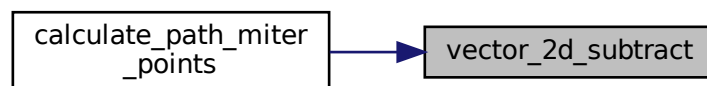


11.4.4.21 `vector_2d_subtract()`

```
void vector_2d_subtract (
    struct vector_2d * res,
    struct vector_2d * a,
    struct vector_2d * b )
```

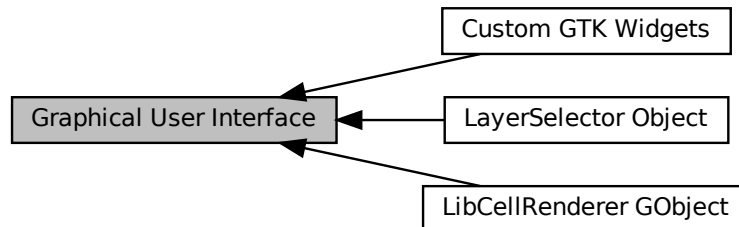
Definition at line 134 of file `vector-operations.c`.

Here is the caller graph for this function:



11.5 Graphical User Interface

Collaboration diagram for Graphical User Interface:



Modules

- [LayerSelector Object](#)
- [LibCellRenderer GObject](#)
- [Custom GTK Widgets](#)

Data Structures

- [struct _GdsRenderGui](#)
- [struct tree_stores](#)

Macros

- `#define RENDERER_TYPE_GUI (gds_render_gui_get_type())`

Enumerations

- `enum gds_render_gui_signal_sig_ids { SIGNAL_WINDOW_CLOSED = 0, SIGNAL_COUNT }`
- `enum cell_store_columns { CELL_SEL_LIBRARY = 0, CELL_SEL_CELL, CELL_SEL_CELL_ERROR_STATE, CELL_SEL_MODDATE, CELL_SEL_ACCESSDATE, CELL_SEL_COLUMN_COUNT }`

Columns of selection tree view.

Functions

- static gboolean [on_window_close](#) (gpointer window, GdkEvent *event, gpointer user)
Main window close event.
- static GString * [generate_string_from_date](#) (struct [gds_time_field](#) *date)
generate string from [gds_time_field](#)
- static void [on_load_gds](#) (gpointer button, gpointer user)
Callback function of Load GDS button.
- static void [on_convert_clicked](#) (gpointer button, gpointer user)
Convert button callback.
- static void [cell_tree_view_activated](#) (gpointer tree_view, GtkTreePath *path, GtkTreeViewColumn *column, gpointer user)
cell_tree_view_activated Callback for 'double click' on cell selector element
- static void [cell_selection_changed](#) (GtkTreeSelection *sel, GdsRenderGui *self)
Callback for cell-selection change event.
- static void [sort_up_callback](#) (GtkWidget *widget, gpointer user)
- static void [sort_down_callback](#) (GtkWidget *widget, gpointer user)
- static void [gds_render_gui_dispose](#) (GObject *gobject)
- static void [gds_render_gui_class_init](#) (GdsRenderGuiClass *klass)
- GtkWidget * [gds_render_gui_get_main_window](#) (GdsRenderGui *gui)
Get main window.
- static void [gds_render_gui_init](#) (GdsRenderGui *self)
- GdsRenderGui * [gds_render_gui_new](#) ()
Create new GdsRenderGui Object.
- G_BEGIN_DECLS [G_DECLARE_FINAL_TYPE](#) (GdsRenderGui, gds_render_gui, RENDERER, GUI, GObject)
- struct [tree_stores](#) * [setup_cell_selector](#) (GtkTreeView *view, GtkEntry *search_entry)
Setup a GtkTreeView with the necessary columns.
- static gboolean [tree_sel_func](#) (GtkTreeSelection *selection, GtkTreeModel *model, GtkTreePath *path, gboolean path_currently_selected, gpointer data)
this function only allows cells to be selected
- static gboolean [cell_store_filter_visible_func](#) (GtkTreeModel *model, GtkTreeIter *iter, gpointer data)
cell_store_filter_visible_func Decides whether an element of the tree model `model` is visible.
- static void [change_filter](#) (GtkWidget *entry, gpointer data)

Variables

- static guint [gds_render_gui_signals](#) [SIGNAL_COUNT]

11.5.1 Detailed Description

11.5.2 Macro Definition Documentation

11.5.2.1 RENDERER_TYPE_GUI

```
#define RENDERER_TYPE_GUI (gds_render_gui_get_type())
```

Definition at line 40 of file [gds-render-gui.h](#).

11.5.3 Enumeration Type Documentation

11.5.3.1 `cell_store_columns`

enum `cell_store_columns`

Columns of selection tree view.

Enumerator

CELL_SEL_LIBRARY	
CELL_SEL_CELL	
CELL_SEL_CELL_ERROR_STATE	Used for cell color and selectability
CELL_SEL_MODDATE	
CELL_SEL_ACCESSDATE	
CELL_SEL_COLUMN_COUNT	Not a column. Used to determine count of columns.

Definition at line 37 of file [tree-store.h](#).

11.5.3.2 `gds_render_gui_signal_sig_ids`

enum `gds_render_gui_signal_sig_ids`

Enumerator

SIGNAL_WINDOW_CLOSED	
SIGNAL_COUNT	

Definition at line 45 of file [gds-render-gui.c](#).

11.5.4 Function Documentation

11.5.4.1 `cell_selection_changed()`

```
static void cell_selection_changed (
    GtkTreeSelection * sel,
    GdsRenderGui * self ) [static]
```

Callback for cell-selection change event.

This function activates/deactivates the convert button depending on whether a cell is selected for conversion or not

Parameters

<i>sel</i>	
<i>self</i>	

Definition at line 397 of file [gds-render-gui.c](#).

Here is the caller graph for this function:



11.5.4.2 cell_store_filter_visible_func()

```

static gboolean cell_store_filter_visible_func (
    GtkTreeModel * model,
    GtkTreeIter * iter,
    gpointer data ) [static]
  
```

`cell_store_filter_visible_func` Decides whether an element of the tree model `model` is visible.

Parameters

<i>model</i>	Tree model
<i>iter</i>	Current element / iter in Model to check
<i>data</i>	Data. Set to static stores variable

Returns

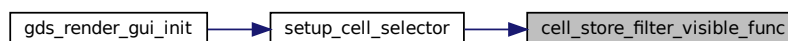
TRUE if visible, else FALSE

Note

TODO: Maybe implement Damerau-Levenshtein distance matching

Definition at line 79 of file [tree-store.c](#).

Here is the caller graph for this function:



11.5.4.3 cell_tree_view_activated()

```
static void cell_tree_view_activated (
    gpointer tree_view,
    GtkTreePath * path,
    GtkTreeViewColumn * column,
    gpointer user ) [static]
```

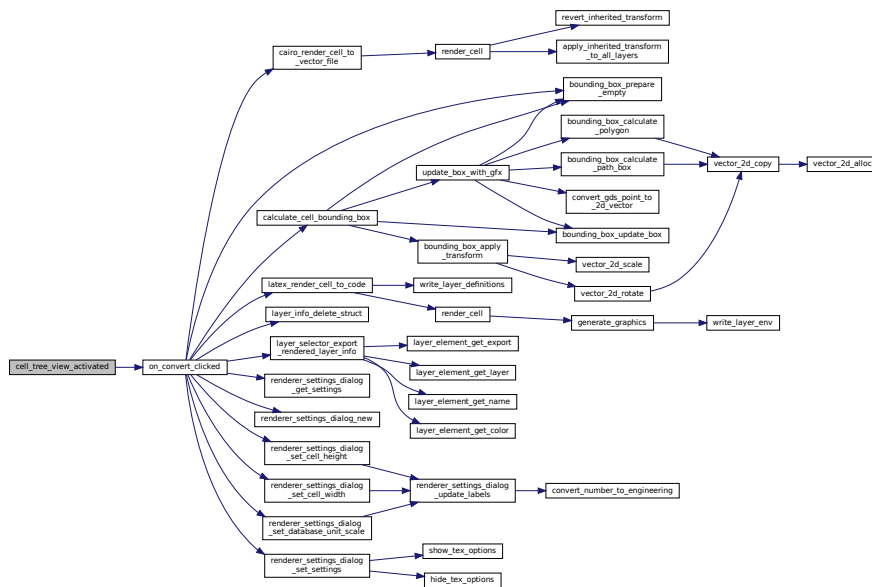
cell_tree_view_activated Callback for 'double click' on cell selector element

Parameters

<i>tree_view</i>	The tree view the event occurred in
<i>path</i>	path to the selected row
<i>column</i>	The clicked column
<i>user</i>	pointer to GdsRenderGui object

Definition at line 378 of file [gds-render-gui.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

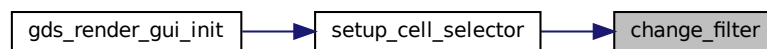


11.5.4.4 `change_filter()`

```
static void change_filter (
    GtkWidget * entry,
    gpointer data ) [static]
```

Definition at line 115 of file [tree-store.c](#).

Here is the caller graph for this function:



11.5.4.5 `G_DECLARE_FINAL_TYPE()`

```
G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (
    GdsRenderGui ,
    gds_render_gui ,
    RENDERER ,
    GUI ,
    GObject )
```

11.5.4.6 `gds_render_gui_class_init()`

```
static void gds_render_gui_class_init (
    GdsRenderGuiClass * klass ) [static]
```

Definition at line 456 of file [gds-render-gui.c](#).

Here is the call graph for this function:

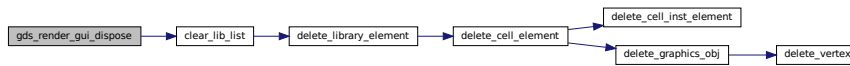


11.5.4.7 `gds_render_gui_dispose()`

```
static void gds_render_gui_dispose (
    GObject * gobject ) [static]
```

Definition at line [432](#) of file [gds-render-gui.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.4.8 `gds_render_gui_get_main_window()`

```
GtkWindow * gds_render_gui_get_main_window (
    GdsRenderGui * gui )
```

Get main window.

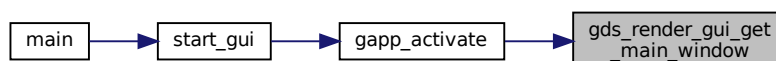
This function returns the main window of the GUI, which can later be displayed. All handling of the GUI is taken care of inside the GdsRenderGui Object

Returns

The generated main window

Definition at line [474](#) of file [gds-render-gui.c](#).

Here is the caller graph for this function:

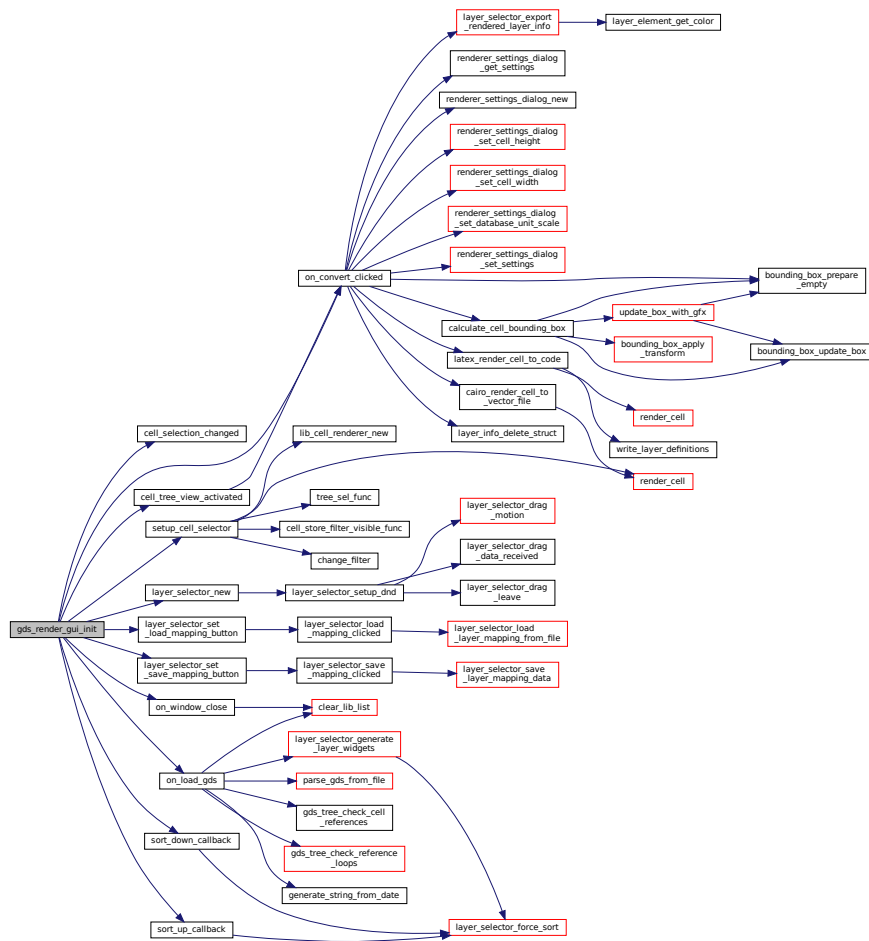


11.5.4.9 gds_render_gui_init()

```
static void gds_render_gui_init (
    GdsRenderGui * self ) [static]
```

Definition at line 479 of file [gds-render-gui.c](#).

Here is the call graph for this function:



11.5.4.10 gds_render_gui_new()

```
GdsRenderGui * gds_render_gui_new ( )
```

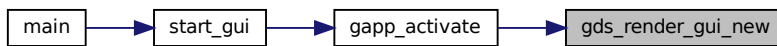
Create new GdsRenderGui Object.

Returns

New object

Definition at line 554 of file [gds-render-gui.c](#).

Here is the caller graph for this function:

**11.5.4.11 generate_string_from_date()**

```
static GString* generate_string_from_date (  
    struct gds\_time\_field * date ) [static]
```

generate string from [gds_time_field](#)

Parameters

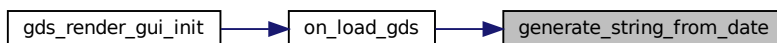
<i>date</i>	Date to convert
-------------	-----------------

Returns

String with date

Definition at line 99 of file [gds-render-gui.c](#).

Here is the caller graph for this function:

**11.5.4.12 on_convert_clicked()**

```
static void on_convert_clicked (  
    gpointer button,  
    gpointer user ) [static]
```

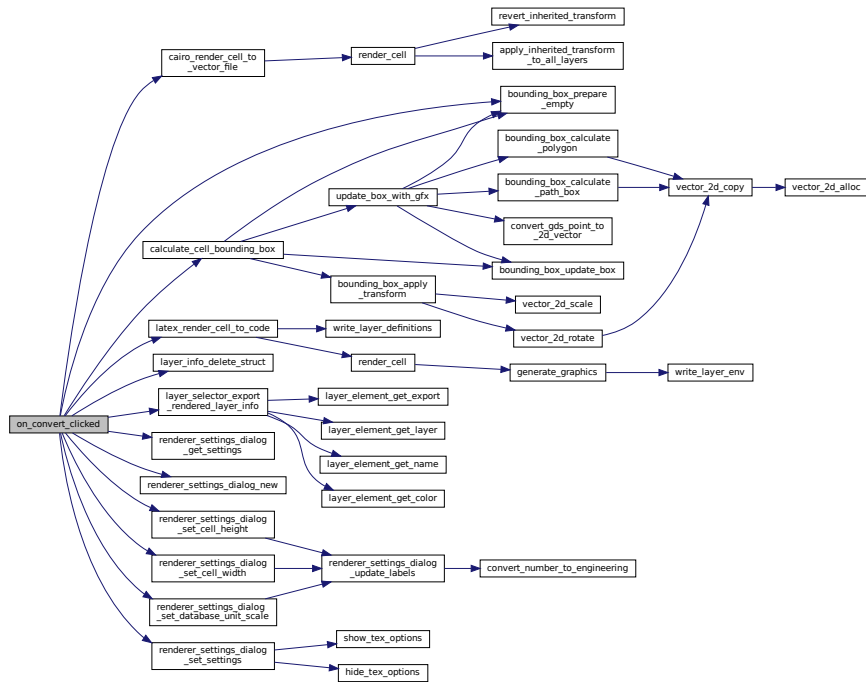
Convert button callback.

Parameters

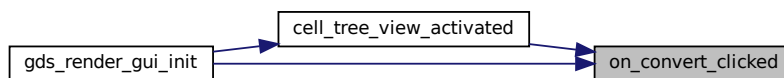
<i>button</i>	
<i>user</i>	

Definition at line 245 of file `gds-render-gui.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.4.13 on_load_gds()

```

static void on_load_gds (
    gpointer button,
    gpointer user ) [static]
  
```

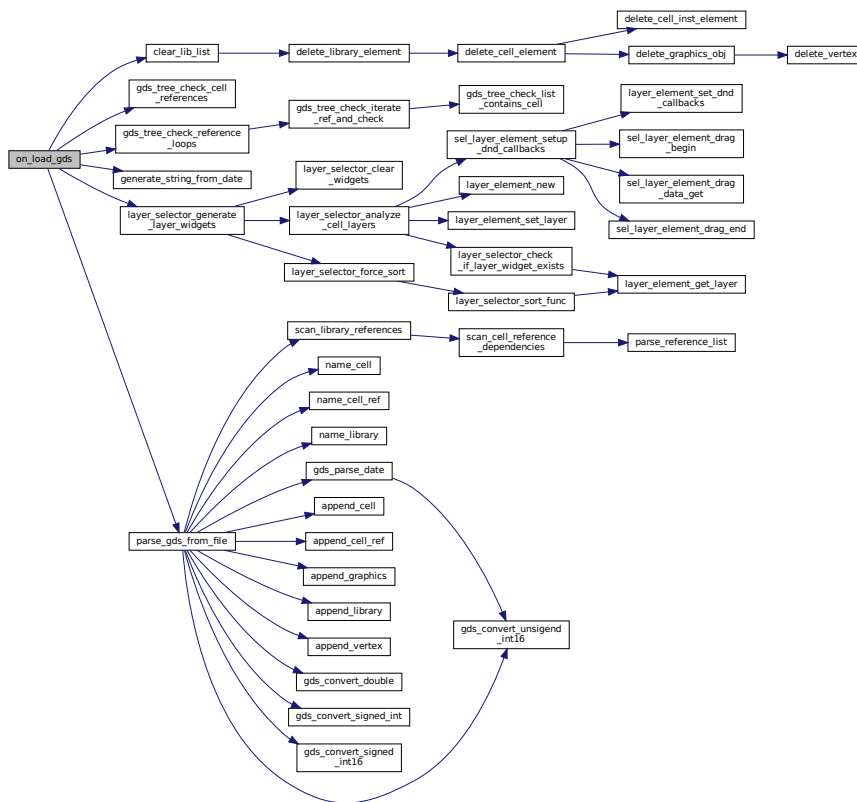
Callback function of Load GDS button.

Parameters

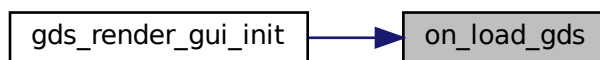
<i>button</i>	
<i>user</i>	GdsRenderGui instance

Definition at line 118 of file [gds-render-gui.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.4.14 on_window_close()

```
static gboolean on_window_close (
    gpointer window,
    GdkEvent * event,
    gpointer user ) [static]
```

Main window close event.

Parameters

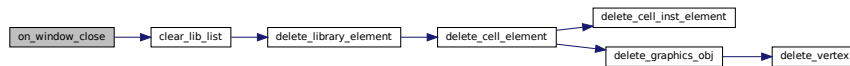
<i>window</i>	GtkWindow which is closed
<i>event</i>	unused event
<i>user</i>	GdsRenderGui instance

Returns

Status of the event handling. Always true.

Definition at line 73 of file [gds-render-gui.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.4.15 setup_cell_selector()

```
struct tree_stores * setup_cell_selector (
    GtkTreeView * view,
    GtkEntry * search_entry )
```

Setup a GtkTreeView with the necessary columns.

Parameters

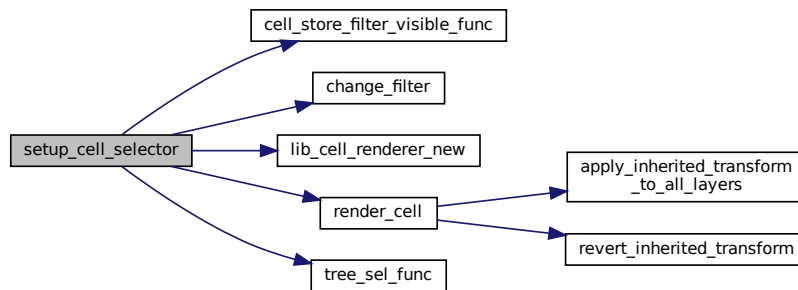
<i>view</i>	Tree view to set up
<i>search_entry</i>	Entry field for search

Returns

Tree stores for storing data inside the GtkTreeView

Definition at line 129 of file [tree-store.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

11.5.4.16 `sort_down_callback()`

```

static void sort_down_callback (
    GtkWidget * widget,
    gpointer user ) [static]
  
```

Definition at line 421 of file [gds-render-gui.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.4.17 sort_up_callback()

```

static void sort_up_callback (
    GtkWidget * widget,
    gpointer user ) [static]
  
```

Definition at line 410 of file [gds-render-gui.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.4.18 tree_sel_func()

```
static gboolean tree_sel_func (
    GtkTreeSelection * selection,
    GtkTreeModel * model,
    GtkTreePath * path,
    gboolean path_currently_selected,
    gpointer data ) [static]
```

this function only allows cells to be selected

Parameters

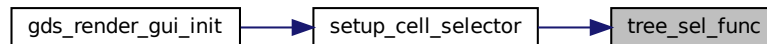
<i>selection</i>	
<i>model</i>	
<i>path</i>	
<i>path_currently_selected</i>	
<i>data</i>	

Returns

TRUE if element is selectable, FALSE if not

Definition at line 44 of file [tree-store.c](#).

Here is the caller graph for this function:



11.5.5 Variable Documentation

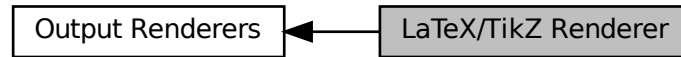
11.5.5.1 gds_render_gui_signals

```
guint gds_render_gui_signals[SIGNAL_COUNT] [static]
```

Definition at line 47 of file [gds-render-gui.c](#).

11.6 LaTeX/TikZ Renderer

Collaboration diagram for LaTeX/TikZ Renderer:



Macros

- `#define LATEX_LINE_BUFFER_KB (10)`
Buffer for LaTeX Code line in KiB.
- `#define WRITEOUT_BUFFER(buff) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)`
*Writes a GString *buffer* to the fixed file *tex_file*.*

Functions

- void `latex_render_cell_to_code` (struct `gds_cell` *cell, GList *layer_infos, FILE *tex_file, double scale, gboolean create_pdf_layers, gboolean standalone_document)
*Render *cell* to LaTeX/TikZ code.*
- static void `write_layer_definitions` (FILE *tex_file, GList *layer_infos, GString *buffer)
Write the layer declarration to TeX file.
- static gboolean `write_layer_env` (FILE *tex_file, GdkRGBA *color, int layer, GList *linfo, GString *buffer)
Write layer Environment.
- static void `generate_graphics` (FILE *tex_file, GList *graphics, GList *linfo, GString *buffer, double scale)
*Writes a graphics object to the specified *tex_file*.*
- static void `render_cell` (struct `gds_cell` *cell, GList *layer_infos, FILE *tex_file, GString *buffer, double scale)
Render cell to file.

11.6.1 Detailed Description

11.6.2 Macro Definition Documentation

11.6.2.1 LATEX_LINE_BUFFER_KB

```
#define LATEX_LINE_BUFFER_KB (10)
```

Buffer for LaTeX Code line in KiB.

Definition at line 40 of file `latex-output.h`.

11.6.2.2 WRITEOUT_BUFFER

```
#define WRITEOUT_BUFFER(  
    buff ) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)
```

Writes a GString `buffer` to the fixed file `tex_file`.

Definition at line 36 of file [latex-output.c](#).

11.6.3 Function Documentation

11.6.3.1 generate_graphics()

```
static void generate_graphics (  
    FILE * tex_file,  
    GList * graphics,  
    GList * linfo,  
    GString * buffer,  
    double scale ) [static]
```

Writes a graphics object to the specified `tex_file`.

This function opens the layer, writes a graphics object and closes the layer

Parameters

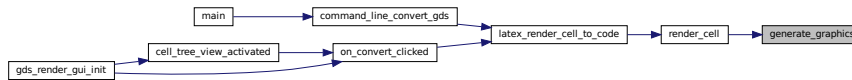
<i>tex_file</i>	File to write to
<i>graphics</i>	Object to render
<i>linfo</i>	Layer information
<i>buffer</i>	Working buffer
<i>scale</i>	Scale object down by this value

Definition at line 138 of file [latex-output.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.6.3.2 latex_render_cell_to_code()

```

void latex_render_cell_to_code (
    struct gds_cell * cell,
    GList * layer_infos,
    FILE * tex_file,
    double scale,
    gboolean create_pdf_layers,
    gboolean standalone_document )
  
```

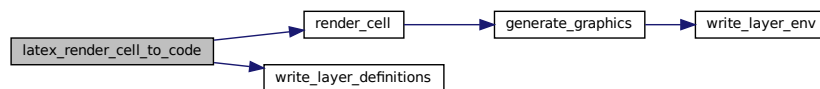
Render `cell` to LateX/TikZ code.

Parameters

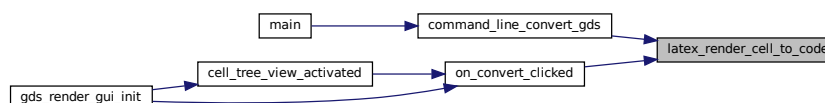
<i>cell</i>	Cell to render
<i>layer_infos</i>	Layer information
<i>tex_file</i>	Already opened file to write data in
<i>scale</i>	Scale image down by this value
<i>create_pdf_layers</i>	Optional content groups used
<i>standalone_document</i>	document can be compiled standalone

Definition at line [252](#) of file [latex-output.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.6.3.3 render_cell()

```
static void render_cell (
    struct gds_cell * cell,
    GList * layer_infos,
    FILE * tex_file,
    GString * buffer,
    double scale ) [static]
```

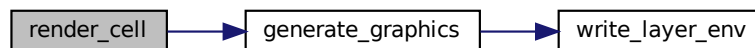
Render cell to file.

Parameters

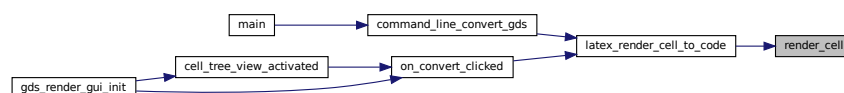
<i>cell</i>	Cell to render
<i>layer_infos</i>	Layer information
<i>tex_file</i>	File to write to
<i>buffer</i>	Working buffer
<i>scale</i>	Scale output down by this value

Definition at line 209 of file [latex-output.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.6.3.4 write_layer_definitions()

```
static void write_layer_definitions (
    FILE * tex_file,
```

```
GList * layer_infos,  
GString * buffer ) [static]
```

Write the layer declaration to TeX file.

This writes the declaration of the layers and the mapping in which order the layers shall be rendered by TikZ. Layers are written in the order they are positioned inside the `layer_infos` list.

Parameters

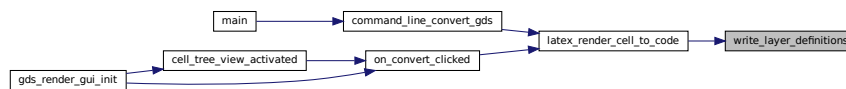
<i>tex_file</i>	TeX-File to write to
<i>layer_infos</i>	List containing layer_info structs.
<i>buffer</i>	

Note

The field [layer_info::stacked_position](#) is ignored. Stack depends on list order.

Definition at line 50 of file [latex-output.c](#).

Here is the caller graph for this function:



11.6.3.5 write_layer_env()

```

static gboolean write_layer_env (
    FILE * tex_file,
    GdkRGBA * color,
    int layer,
    GList * linfo,
    GString * buffer ) [static]
  
```

Write layer Environment.

If the requested layer shall be rendered, this code writes the necessary code to open the layer. It also returns the color the layer shall be rendered in.

The following environments are generated:

```

\begin{pgfonlayer}{<layer>}
% If pdf layers shall be used also this is enabled:
\begin{scope}[ocg={ref=<layer>, status=visible,name={<Layer Name>}}]
  
```

If the layer shall not be rendered, FALSE is returned and the color is not filled in and the code is not written to the file.

Parameters

<i>tex_file</i>	TeX file to write to
<i>color</i>	Return of the layer's color
<i>layer</i>	Requested layer number
<i>linfo</i>	Layer information list containing layer_info structs
<i>buffer</i>	Some working buffer

Returns

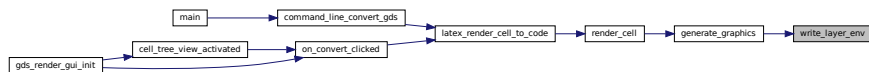
TRUE, if the layer shall be rendered.

Note

The opened environments have to be closed afterwards

Definition at line 106 of file [latex-output.c](#).

Here is the caller graph for this function:



11.7 LayerSelector Object

Collaboration diagram for LayerSelector Object:



Data Structures

- struct [_LayerSelector](#)

Macros

- #define [TYPE_LAYER_SELECTOR](#) ([layer_selector_get_type\(\)](#))

Enumerations

- enum [layer_selector_sort_algo](#) { [LAYER_SELECTOR_SORT_DOWN](#) = 0, [LAYER_SELECTOR_SORT_UP](#) }

Defines how to sort the layer selector list box.

Functions

- G_BEGIN_DECLS [G_DECLARE_FINAL_TYPE](#) (LayerSelector, layer_selector, [LAYER_SELECTOR](#), G↳ Object)
- LayerSelector * [layer_selector_new](#) (GtkListBox *list_box)
layer_selector_new
- void [layer_selector_generate_layer_widgets](#) (LayerSelector *selector, GList *libs)
Generate layer widgets in in the LayerSelector instance.
- void [layer_selector_set_load_mapping_button](#) (LayerSelector *selector, GtkWidget *button, GtkWidget *main_window)
Supply button for loading the layer mapping.
- void [layer_selector_set_save_mapping_button](#) (LayerSelector *selector, GtkWidget *button, GtkWidget *main_window)
Supply button for saving the layer mapping.
- GList * [layer_selector_export_rendered_layer_info](#) (LayerSelector *selector)
Get a list of all layers that shall be exported when rendering the cells.
- void [layer_selector_force_sort](#) (LayerSelector *selector, enum [layer_selector_sort_algo](#) sort_function)
*Force the layer selector list to be sorted according to *sort_function*.*
- static void [sel_layer_element_drag_begin](#) (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void [sel_layer_element_drag_end](#) (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void [sel_layer_element_drag_data_get](#) (GtkWidget *widget, GdkDragContext *context, GtkSelection↳ Data *selection_data, guint info, guint time, gpointer data)

- static GtkWidget * [layer_selector_get_last_row](#) (GtkWidget *list)
- static GtkWidget * [layer_selector_get_row_before](#) (GtkWidget *list, GtkWidget *row)
- static GtkWidget * [layer_selector_get_row_after](#) (GtkWidget *list, GtkWidget *row)
- static void [layer_selector_drag_data_received](#) (GtkWidget *widget, GdkDragContext *context, gint x, gint y, GtkSelectionData *selection_data, guint info, guint32 time, gpointer data)
- static gboolean [layer_selector_drag_motion](#) (GtkWidget *widget, GdkDragContext *context, int x, int y, guint time)
- static void [layer_selector_drag_leave](#) (GtkWidget *widget, GdkDragContext *context, guint time)
- static void [layer_selector_dispose](#) (GObject *self)
- static void [layer_selector_class_init](#) (LayerSelectorClass *klass)
- static void [layer_selector_setup_dnd](#) (LayerSelector *self)
- static void [layer_selector_init](#) (LayerSelector *self)
- static void [layer_selector_clear_widgets](#) (LayerSelector *self)
- static gboolean [layer_selector_check_if_layer_widget_exists](#) (LayerSelector *self, int layer)

Check if a specific layer element with the given layer number is present in the layer selector.
- static void [sel_layer_element_setup_dnd_callbacks](#) (LayerSelector *self, LayerElement *element)

Setup the necessary drag and drop callbacks of layer elements.
- static void [layer_selector_analyze_cell_layers](#) (LayerSelector *self, struct [gds_cell](#) *cell)

Analyze `cell` layers and append detected layers to layer selector `self`.
- static gint [layer_selector_sort_func](#) (GtkWidget *row1, GtkWidget *row2, gpointer unused)

sort_func Sort callback for list box
- static LayerElement * [layer_selector_find_layer_element_in_list](#) (GList *el_list, int layer)

Find LayerElement in list with specified layer number.
- static void [layer_selector_load_layer_mapping_from_file](#) (LayerSelector *self, gchar *file_name)

Load the layer mapping from a CSV formatted file.
- static void [layer_selector_load_mapping_clicked](#) (GtkWidget *button, gpointer user_data)

Callback for Load Mapping Button.
- static void [layer_selector_save_layer_mapping_data](#) (LayerSelector *self, const gchar *file_name)

Save layer mapping of selector `self` to a file.
- static void [layer_selector_save_mapping_clicked](#) (GtkWidget *button, gpointer user_data)

Callback for Save Layer Mapping Button.

Variables

- static const char * [dnd_additional_css](#)

11.7.1 Detailed Description

This objects implements the layer selector and displays the layers in a list box. It uses [LayerElement](#) objects to display the individual layers inside the list box.

11.7.2 Macro Definition Documentation

11.7.2.1 TYPE_LAYER_SELECTOR

```
#define TYPE_LAYER_SELECTOR (layer_selector_get_type())
```

Definition at line 41 of file [layer-selector.h](#).

11.7.3 Enumeration Type Documentation

11.7.3.1 layer_selector_sort_algo

```
enum layer_selector_sort_algo
```

Defines how to sort the layer selector list box.

Enumerator

LAYER_SELECTOR_SORT_DOWN	
LAYER_SELECTOR_SORT_UP	

Definition at line 46 of file [layer-selector.h](#).

11.7.4 Function Documentation

11.7.4.1 G_DECLARE_FINAL_TYPE()

```
G_BEGIN_DECLS G_DECLARE_FINAL_TYPE (
    LayerSelector ,
    layer_selector ,
    LAYER ,
    SELECTOR ,
    GObject )
```

11.7.4.2 layer_selector_analyze_cell_layers()

```
static void layer_selector_analyze_cell_layers (
    LayerSelector * self,
    struct gds_cell * cell ) [static]
```

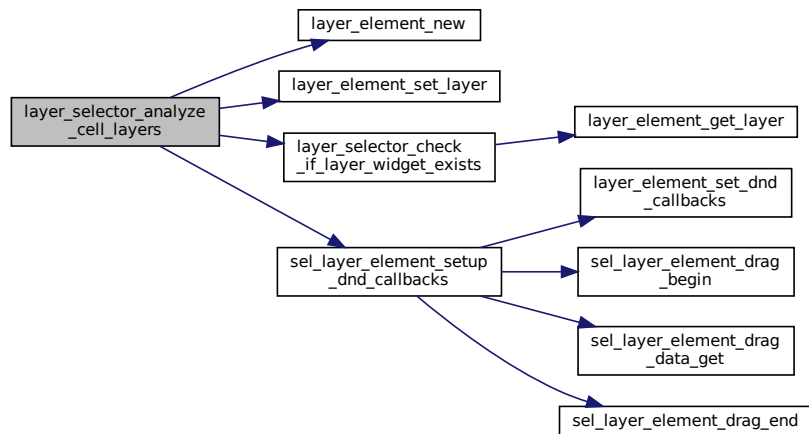
Analyze `cell` layers and append detected layers to layer selector `self`.

Parameters

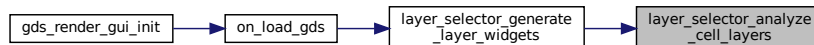
<i>self</i>	LayerSelector instance
<i>cell</i>	Cell to analyze

Definition at line 482 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.3 layer_selector_check_if_layer_widget_exists()

```

static gboolean layer_selector_check_if_layer_widget_exists (
    LayerSelector * self,
    int layer ) [static]
  
```

Check if a specific layer element with the given layer number is present in the layer selector.

Parameters

<i>self</i>	LayerSelector instance
<i>layer</i>	Layer number to check for

Returns

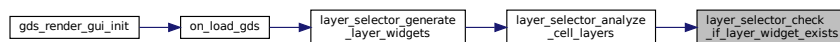
TRUE if layer is present, else FALSE

Definition at line 435 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.4 layer_selector_class_init()

```
static void layer_selector_class_init (
    LayerSelectorClass * klass ) [static]
```

Definition at line 321 of file [layer-selector.c](#).

Here is the call graph for this function:

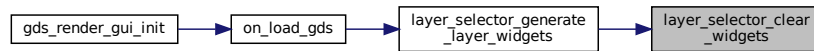


11.7.4.5 layer_selector_clear_widgets()

```
static void layer_selector_clear_widgets (
    LayerSelector * self ) [static]
```

Definition at line 409 of file [layer-selector.c](#).

Here is the caller graph for this function:



11.7.4.6 layer_selector_dispose()

```
static void layer_selector_dispose (
    GObject * self ) [static]
```

Definition at line [302](#) of file [layer-selector.c](#).

Here is the caller graph for this function:

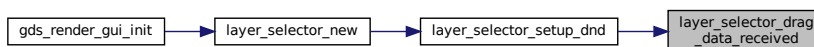


11.7.4.7 layer_selector_drag_data_received()

```
static void layer_selector_drag_data_received (
    GtkWidget * widget,
    GdkDragContext * context,
    gint x,
    gint y,
    GtkSelectionData * selection_data,
    guint info,
    guint32 time,
    gpointer data ) [static]
```

Definition at line [153](#) of file [layer-selector.c](#).

Here is the caller graph for this function:

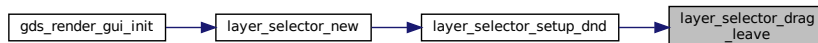


11.7.4.8 layer_selector_drag_leave()

```
static void layer_selector_drag_leave (  
    GtkWidget * widget,  
    GdkDragContext * context,  
    guint time ) [static]
```

Definition at line 248 of file [layer-selector.c](#).

Here is the caller graph for this function:

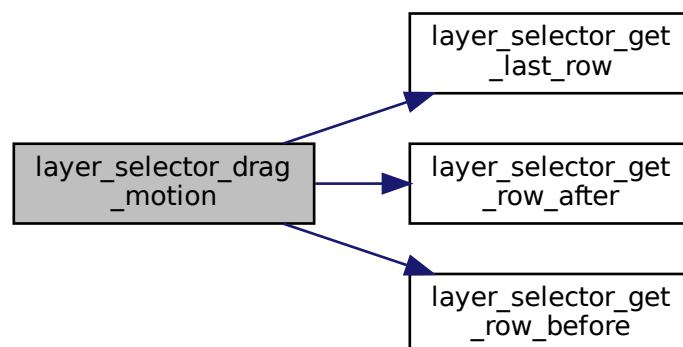


11.7.4.9 layer_selector_drag_motion()

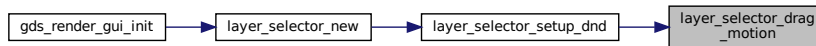
```
static gboolean layer_selector_drag_motion (  
    GtkWidget * widget,  
    GdkDragContext * context,  
    int x,  
    int y,  
    guint time ) [static]
```

Definition at line 191 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.10 layer_selector_export_rendered_layer_info()

```

GList * layer_selector_export_rendered_layer_info (
    LayerSelector * selector )
  
```

Get a list of all layers that shall be exported when rendering the cells.

Parameters

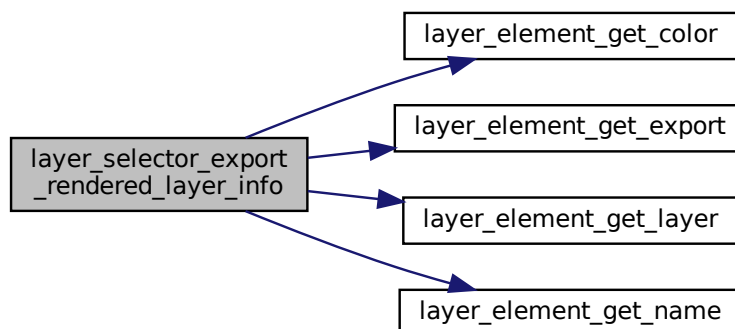
<i>selector</i>	Layer selector instance
-----------------	-------------------------

Returns

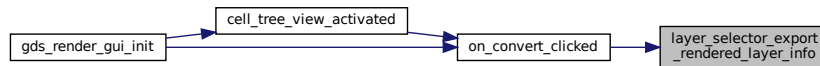
List of [layer_info](#) structures containing the layer information

Definition at line [374](#) of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.11 layer_selector_find_layer_element_in_list()

```

static LayerElement* layer_selector_find_layer_element_in_list (
    GList * el_list,
    int layer ) [static]
  
```

Find LayerElement in list with specified layer number.

Parameters

<i>el_list</i>	List with elements of type LayerElement
<i>layer</i>	Layer number

Returns

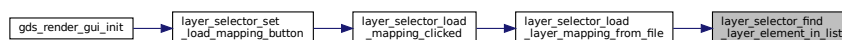
Found LayerElement. If nothing is found, NULL.

Definition at line [564](#) of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.12 layer_selector_force_sort()

```
void layer_selector_force_sort (
    LayerSelector * selector,
    enum layer_selector_sort_algo sort_function )
```

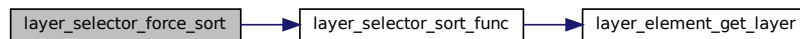
Force the layer selector list to be sorted according to `sort_function`.

Parameters

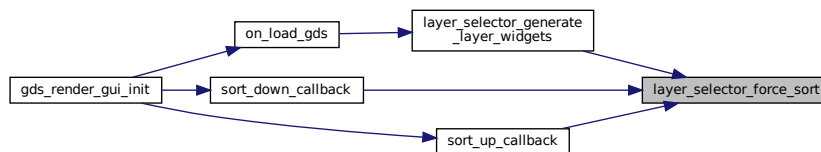
<i>selector</i>	LayerSelector instance
<i>sort_function</i>	The sorting method (up or down sorting)

Definition at line 767 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.13 layer_selector_generate_layer_widgets()

```
void layer_selector_generate_layer_widgets (
    LayerSelector * selector,
    GList * libs )
```

Generate layer widgets in in the LayerSelector instance.

Note

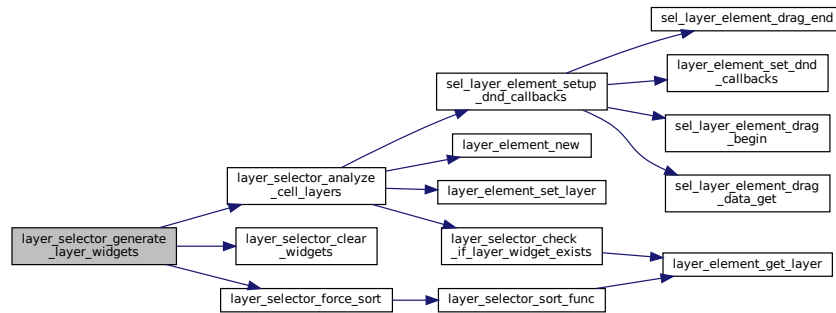
This clears all previously inserted elements

Parameters

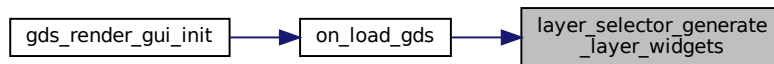
<i>selector</i>	LayerSelector instance
<i>libs</i>	The libraries to add

Definition at line 534 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

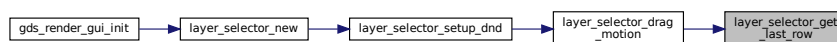


11.7.4.14 layer_selector_get_last_row()

```
static GtkWidget* layer_selector_get_last_row (
    GtkWidget * list ) [static]
```

Definition at line 120 of file [layer-selector.c](#).

Here is the caller graph for this function:

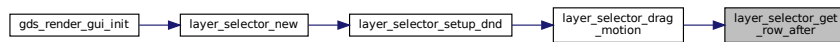


11.7.4.15 `layer_selector_get_row_after()`

```
static GtkWidgetRow* layer_selector_get_row_after (
    GtkWidget * list,
    GtkWidgetRow * row ) [static]
```

Definition at line 145 of file [layer-selector.c](#).

Here is the caller graph for this function:

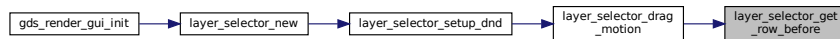


11.7.4.16 `layer_selector_get_row_before()`

```
static GtkWidgetRow* layer_selector_get_row_before (
    GtkWidget * list,
    GtkWidgetRow * row ) [static]
```

Definition at line 137 of file [layer-selector.c](#).

Here is the caller graph for this function:



11.7.4.17 `layer_selector_init()`

```
static void layer_selector_init (
    LayerSelector * self ) [static]
```

Definition at line 347 of file [layer-selector.c](#).

11.7.4.18 `layer_selector_load_layer_mapping_from_file()`

```
static void layer_selector_load_layer_mapping_from_file (
    LayerSelector * self,
    gchar * file_name ) [static]
```

Load the layer mapping from a CSV formatted file.

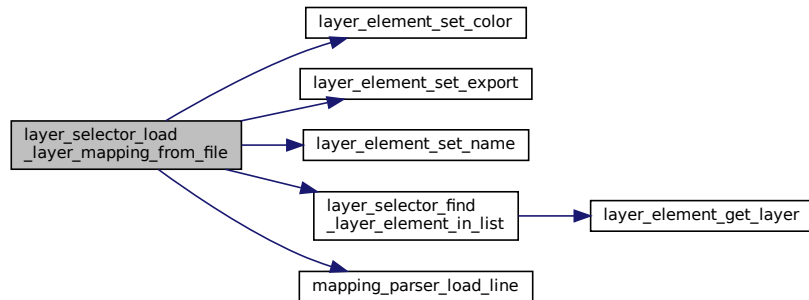
This function imports the layer specification from a file (see [Layer Mapping File Specification](#)). The layer ordering defined in the file is kept. All layers present in the current loaded library, which are not present in the layer mapping file are appended at the end of the layer selector list.

Parameters

<i>self</i>	LayerSelector instance
<i>file_name</i>	File name to load from

Definition at line 587 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.19 layer_selector_load_mapping_clicked()

```

static void layer_selector_load_mapping_clicked (
    GtkWidget * button,
    gpointer user_data ) [static]
  
```

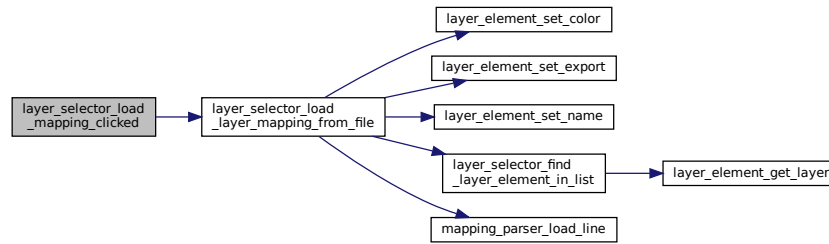
Callback for Load Mapping Button.

Parameters

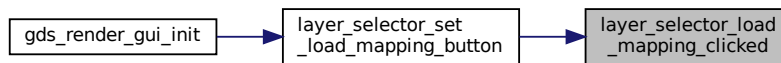
<i>button</i>	
<i>user_data</i>	

Definition at line 662 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.20 layer_selector_new()

```

LayerSelector * layer_selector_new (
    GtkWidget * list_box )
  
```

`layer_selector_new`

Parameters

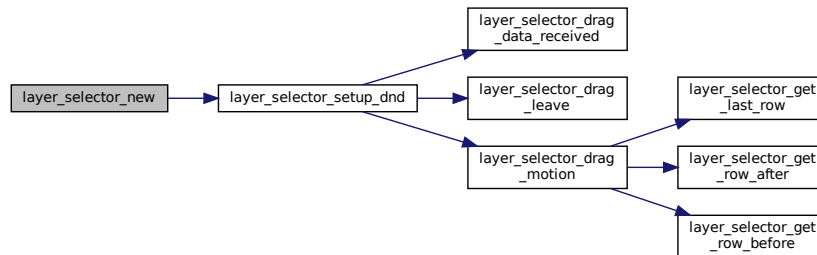
<code>list_box</code>	The associated list box, the content is displayed in
-----------------------	--

Returns

Newly created layer selector

Definition at line 359 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**11.7.4.21 layer_selector_save_layer_mapping_data()**

```

static void layer_selector_save_layer_mapping_data (
    LayerSelector * self,
    const gchar * file_name ) [static]
  
```

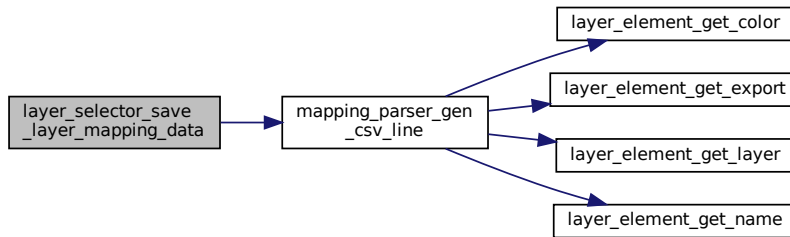
Save layer mapping of selector `self` to a file.

Parameters

<i>self</i>	LayerSelector instance
<i>file_name</i>	File name to save to

Definition at line 689 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.22 layer_selector_save_mapping_clicked()

```

static void layer_selector_save_mapping_clicked (
    GtkWidget * button,
    gpointer user_data ) [static]
  
```

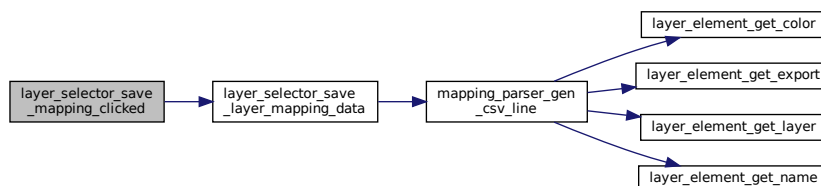
Callback for Save Layer Mapping Button.

Parameters

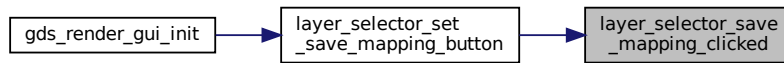
<i>button</i>	
<i>user_data</i>	

Definition at line 721 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.23 layer_selector_set_load_mapping_button()

```

void layer_selector_set_load_mapping_button (
    LayerSelector * selector,
    GtkWidget * button,
    GtkWindow * main_window )
  
```

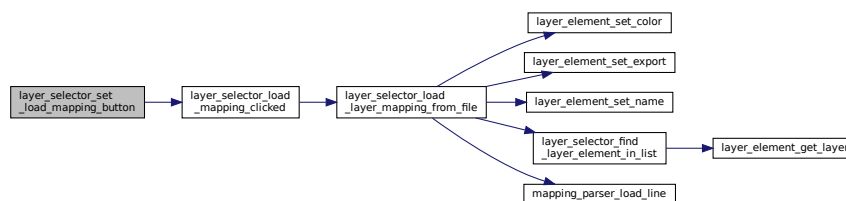
Supply button for loading the layer mapping.

Parameters

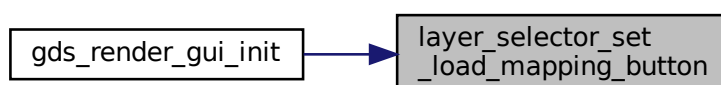
<i>selector</i>	LayerSelector instance
<i>button</i>	Load button. Will be referenced
<i>main_window</i>	Parent window for dialogs. Will be referenced

Definition at line [743](#) of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.24 `layer_selector_set_save_mapping_button()`

```
void layer_selector_set_save_mapping_button (
    LayerSelector * selector,
    GtkWidget * button,
    GtkWindow * main_window )
```

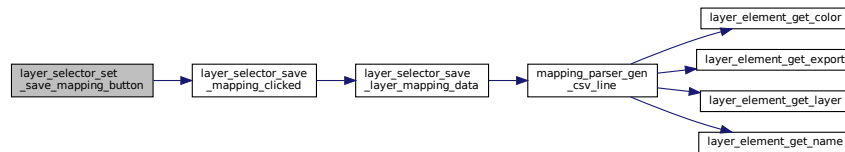
Supply button for saving the layer mapping.

Parameters

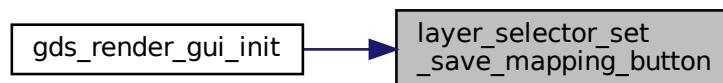
<i>selector</i>	LayerSelector instance
<i>button</i>	Save button. Will be referenced
<i>main_window</i>	Parent window for dialogs. Will be referenced

Definition at line [755](#) of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

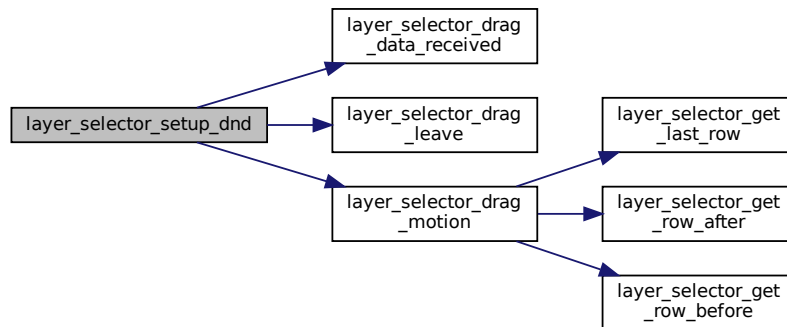


11.7.4.25 `layer_selector_setup_dnd()`

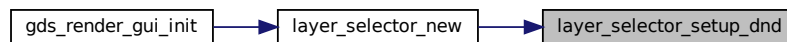
```
static void layer_selector_setup_dnd (
    LayerSelector * self ) [static]
```

Definition at line [337](#) of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.4.26 layer_selector_sort_func()

```

static gint layer_selector_sort_func (
    GtkWidgetRow * row1,
    GtkWidgetRow * row2,
    gpointer unused ) [static]
  
```

`sort_func` Sort callback for list box

Parameters

<i>row1</i>	
<i>row2</i>	
<i>unused</i>	

Note

Do not use this function. This is an internal callback

Returns

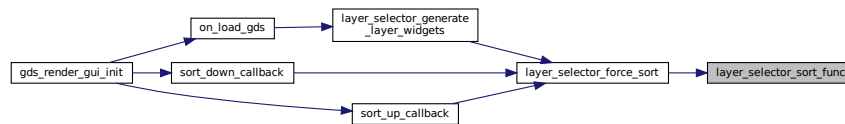
See sort function documentation of GTK+

Definition at line 510 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

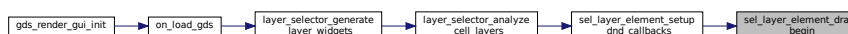
**11.7.4.27 sel_layer_element_drag_begin()**

```

static void sel_layer_element_drag_begin (
    GtkWidget * widget,
    GdkDragContext * context,
    gpointer data ) [static]
  
```

Definition at line 63 of file [layer-selector.c](#).

Here is the caller graph for this function:



11.7.4.28 sel_layer_element_drag_data_get()

```
static void sel_layer_element_drag_data_get (
    GtkWidget * widget,
    GdkDragContext * context,
    GtkSelectionData * selection_data,
    guint info,
    guint time,
    gpointer data ) [static]
```

Definition at line 104 of file [layer-selector.c](#).

Here is the caller graph for this function:

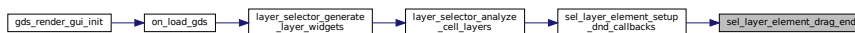


11.7.4.29 sel_layer_element_drag_end()

```
static void sel_layer_element_drag_end (
    GtkWidget * widget,
    GdkDragContext * context,
    gpointer data ) [static]
```

Definition at line 92 of file [layer-selector.c](#).

Here is the caller graph for this function:



11.7.4.30 sel_layer_element_setup_dnd_callbacks()

```
static void sel_layer_element_setup_dnd_callbacks (
    LayerSelector * self,
    LayerElement * element ) [static]
```

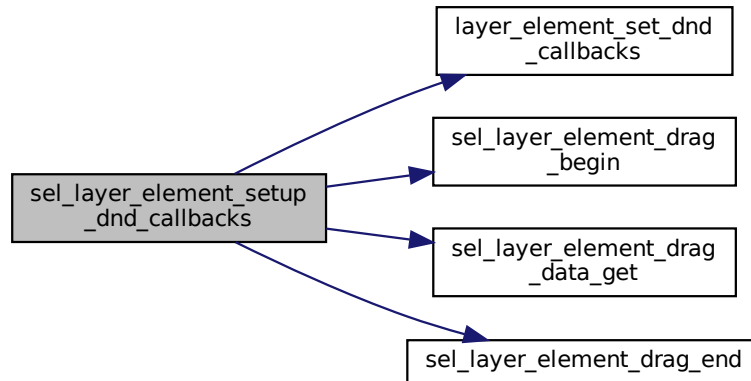
Setup the necessary drag and drop callbacks of layer elements.

Parameters

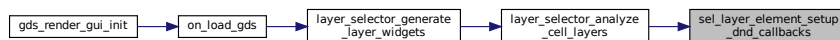
<i>self</i>	LayerSelector instance. Used to get the DnD target entry.
<i>element</i>	LayerElement instance to set the callbacks

Definition at line 461 of file [layer-selector.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.7.5 Variable Documentation

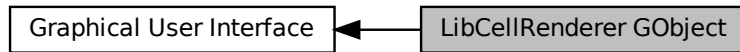
11.7.5.1 dnd_additional_css

```
const char* dnd_additional_css [static]
```

Definition at line 266 of file [layer-selector.c](#).

11.8 LibCellRenderer GObject

Collaboration diagram for LibCellRenderer GObject:



Data Structures

- struct [_LibCellRenderer](#)

Macros

- #define [TYPE_LIB_CELL_RENDERER](#) ([lib_cell_renderer_get_type\(\)](#))
- #define [LIB_CELL_RENDERER_ERROR_WARN](#) (1U<<0)
- #define [LIB_CELL_RENDERER_ERROR_ERR](#) (1U<<1)

Typedefs

- typedef struct [_LibCellRenderer](#) [LibCellRenderer](#)

Enumerations

- enum { [PROP_LIB](#) = 1, [PROP_CELL](#), [PROP_ERROR_LEVEL](#), [PROP_COUNT](#) }

Functions

- GType [lib_cell_renderer_get_type](#) (void)
lib_cell_renderer_get_type
- GtkWidget * [lib_cell_renderer_new](#) (void)
Create a new renderer for rendering [gds_cell](#) and [gds_library](#) elements.
- void [lib_cell_renderer_init](#) ([LibCellRenderer](#) *self)
- static void [lib_cell_renderer_constructed](#) (GObject *obj)
- static void [convert_error_level_to_color](#) (GdkRGBA *color, unsigned int error_level)
- static void [lib_cell_renderer_set_property](#) (GObject *object, guint param_id, const GValue *value, GParamSpec *pspec)
- static void [lib_cell_renderer_get_property](#) (GObject *object, guint param_id, GValue *value, GParamSpec *pspec)
- void [lib_cell_renderer_class_init](#) ([LibCellRendererClass](#) *klass)

Variables

- static GParamSpec * [properties](#) [[PROP_COUNT](#)]

11.8.1 Detailed Description

The LibCellRenderer Object is used to render [gds_cell](#) and [gds_library](#) elements to a GtkTreeView.

The LibCellRenderer class is derived from a GtkCellRendererText and works the same way. The additional features are three new properties:

- *gds-lib*: This property can be used to set a [gds_library](#) structure. The renderer will render the name of the library.
- *gds-cell*: This property can be used to set a [gds_cell](#) structure. The renderer will render the name of the cell.
- *error-level*: Set the error level of the cell/library. This affects the foreground color of the rendered output.

Internally the class operates by setting the 'text' property, which is inherited from the base class to the library/cell name ([gds_library::name](#) and [gds_cell::name](#) fields). The error level ([LIB_CELL_RENDERER_ERROR_WARN](#) and [LIB_CELL_RENDERER_ERROR_ERR](#)) is translated to the inherited 'foreground-rgba' property.

11.8.2 Macro Definition Documentation

11.8.2.1 LIB_CELL_RENDERER_ERROR_ERR

```
#define LIB_CELL_RENDERER_ERROR_ERR (1U<<1)
```

Definition at line 45 of file [lib-cell-renderer.h](#).

11.8.2.2 LIB_CELL_RENDERER_ERROR_WARN

```
#define LIB_CELL_RENDERER_ERROR_WARN (1U<<0)
```

Error levels

Definition at line 44 of file [lib-cell-renderer.h](#).

11.8.2.3 TYPE_LIB_CELL_RENDERER

```
#define TYPE_LIB_CELL_RENDERER (lib_cell_renderer_get_type())
```

Definition at line 39 of file [lib-cell-renderer.h](#).

11.8.3 Typedef Documentation

11.8.3.1 LibCellRenderer

```
typedef struct _LibCellRenderer LibCellRenderer
```

11.8.4 Enumeration Type Documentation

11.8.4.1 anonymous enum

```
anonymous enum
```

Enumerator

PROP_LIB	
PROP_CELL	
PROP_ERROR_LEVEL	
PROP_COUNT	

Definition at line 36 of file [lib-cell-renderer.c](#).

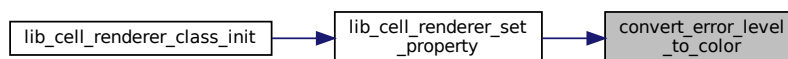
11.8.5 Function Documentation

11.8.5.1 convert_error_level_to_color()

```
static void convert_error_level_to_color (  
    GdkRGBA * color,  
    unsigned int error_level ) [static]
```

Definition at line 53 of file [lib-cell-renderer.c](#).

Here is the caller graph for this function:

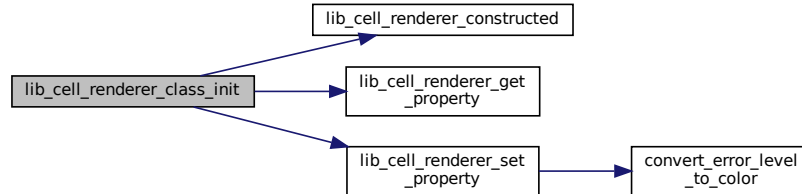


11.8.5.2 lib_cell_renderer_class_init()

```
void lib_cell_renderer_class_init (
    LibCellRendererClass * klass )
```

Definition at line 123 of file [lib-cell-renderer.c](#).

Here is the call graph for this function:



11.8.5.3 lib_cell_renderer_constructed()

```
static void lib_cell_renderer_constructed (
    GObject * obj ) [static]
```

Definition at line 48 of file [lib-cell-renderer.c](#).

Here is the caller graph for this function:

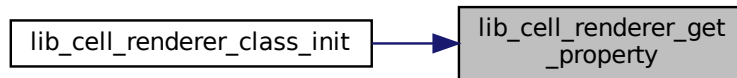


11.8.5.4 lib_cell_renderer_get_property()

```
static void lib_cell_renderer_get_property (
    GObject * object,
    guint param_id,
    GValue * value,
    GParamSpec * pspec ) [static]
```

Definition at line 109 of file [lib-cell-renderer.c](#).

Here is the caller graph for this function:



11.8.5.5 `lib_cell_renderer_get_type()`

```
GType lib_cell_renderer_get_type (
    void )
```

`lib_cell_renderer_get_type`

Returns

GObject Type

11.8.5.6 `lib_cell_renderer_init()`

```
void lib_cell_renderer_init (
    LibCellRenderer * self )
```

Definition at line 43 of file `lib-cell-renderer.c`.

11.8.5.7 `lib_cell_renderer_new()`

```
GtkCellRenderer * lib_cell_renderer_new (
    void )
```

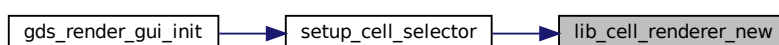
Create a new renderer for rendering `gds_cell` and `gds_library` elements.

Returns

New renderer object

Definition at line 143 of file `lib-cell-renderer.c`.

Here is the caller graph for this function:

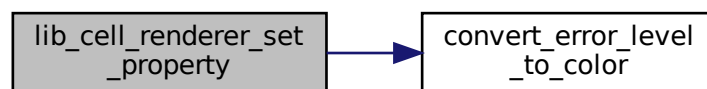


11.8.5.8 lib_cell_renderer_set_property()

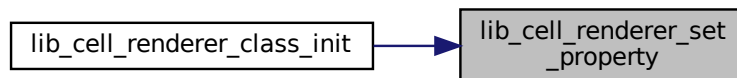
```
static void lib_cell_renderer_set_property (
    GObject * object,
    guint param_id,
    const GValue * value,
    GParamSpec * pspec ) [static]
```

Definition at line 77 of file [lib-cell-renderer.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.8.6 Variable Documentation

11.8.6.1 properties

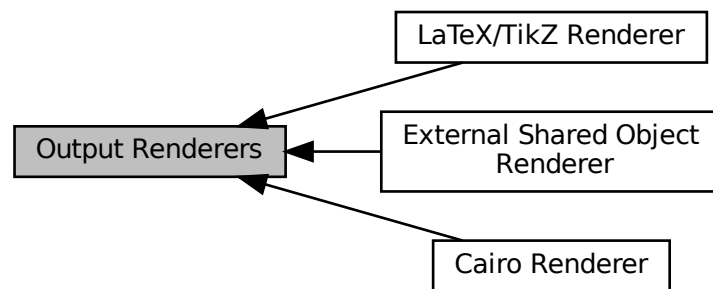
```
GParamSpec* properties[PROP_COUNT] [static]
```

Definition at line 121 of file [lib-cell-renderer.c](#).

11.9 Output Renderers

The renderers are used to convert the cell structures read from the GDS layout file into different output formats.

Collaboration diagram for Output Renderers:



Modules

- [Cairo Renderer](#)
- [External Shared Object Renderer](#)
- [LaTeX/TikZ Renderer](#)

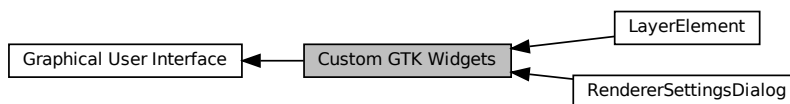
11.9.1 Detailed Description

The renderers are used to convert the cell structures read from the GDS layout file into different output formats.

Currently the renders are statically implemented without the use of GObjects. This will probably change in future releases in order to make it easier to integrate new rendering methods.

11.10 Custom GTK Widgets

Collaboration diagram for Custom GTK Widgets:



Modules

- [RendererSettingsDialog](#)
- [LayerElement](#)

11.10.1 Detailed Description

11.11 GDS-Utilities

Data Structures

- struct `gds_point`
A point in the 2D plane. Sometimes references as vertex.
- struct `gds_cell_checks`
Stores the result of the cell checks.
- struct `gds_time_field`
Date information for cells and libraries.
- struct `gds_graphics`
A GDS graphics object.
- struct `gds_cell_instance`
This represents an instanc of a cell inside another cell.
- struct `gds_cell`
A Cell inside a `gds_library`.
- struct `gds_library`
GDS Toplevel library.

Macros

- #define `GDS_DEFAULT_UNITS` (10E-9)
Default units assumed for library.
- #define `GDS_ERROR`(fmt, ...) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)
Print GDS error.
- #define `GDS_WARN`(fmt, ...) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)
Print GDS warning.
- #define `GDS_INF`(fmt, ...)
- #define `GDS_PRINT_DEBUG_INFOS` (0)
1: Print infos, 0: Don't print
- #define `CELL_NAME_MAX` (100)
Maximum length of a `gds_cell::name` or a `gds_library::name`.
- #define `MIN`(a, b) (((a) < (b)) ? (a) : (b))
Return smaller number.
- #define `MAX`(a, b) (((a) > (b)) ? (a) : (b))
Return bigger number.

Enumerations

- enum `gds_record` {
`INVALID` = 0x0000, `HEADER` = 0x0002, `BGNLIB` = 0x0102, `LIBNAME` = 0x0206,
`UNITS` = 0x0305, `ENDLIB` = 0x0400, `BGNSTR` = 0x0502, `STRNAME` = 0x0606,
`ENDSTR` = 0x0700, `BOUNDARY` = 0x0800, `PATH` = 0x0900, `SREF` = 0x0A00,
`ENDEL` = 0x1100, `XY` = 0x1003, `MAG` = 0x1B05, `ANGLE` = 0x1C05,
`SNAME` = 0x1206, `STRANS` = 0x1A01, `BOX` = 0x2D00, `LAYER` = 0x0D02,
`WIDTH` = 0x0F03, `PATHTYPE` = 0x2102 }
- enum { `GDS_CELL_CHECK_NOT_RUN` = -1 }
- enum `graphics_type` { `GRAPHIC_PATH` = 0, `GRAPHIC_POLYGON` = 1, `GRAPHIC_BOX` = 2 }
Types of graphic objects.
- enum `path_type` { `PATH_FLUSH` = 0, `PATH_ROUNDED` = 1, `PATH_SQUARED` = 2 }
Defines the line caps of a path.

Functions

- static int `name_cell_ref` (struct `gds_cell_instance` *cell_inst, unsigned int bytes, char *data)
Name cell reference.
- static double `gds_convert_double` (const char *data)
Convert GDS 8-byte real to double.
- static signed int `gds_convert_signed_int` (const char *data)
Convert GDS INT32 to int.
- static int16_t `gds_convert_signed_int16` (const char *data)
Convert GDS INT16 to int16.
- static uint16_t `gds_convert_unsigned_int16` (const char *data)
Convert GDS UINT16 String to uint16.
- static GList * `append_library` (GList *curr_list, struct `gds_library` **library_ptr)
Append library to list.
- static GList * `append_graphics` (GList *curr_list, enum `graphics_type` type, struct `gds_graphics` **graphics_ptr)
Append graphics to list.
- static GList * `append_vertex` (GList *curr_list, int x, int y)
Appends vertex List.
- static GList * `append_cell` (GList *curr_list, struct `gds_cell` **cell_ptr)
append_cell Append a gds_cell to a list
- static GList * `append_cell_ref` (GList *curr_list, struct `gds_cell_instance` **instance_ptr)
Append a cell reference to the reference GList.
- static int `name_library` (struct `gds_library` *current_library, unsigned int bytes, char *data)
Name a gds_library.
- static int `name_cell` (struct `gds_cell` *cell, unsigned int bytes, char *data, struct `gds_library` *lib)
Names a gds_cell.
- static void `parse_reference_list` (gpointer gcell_ref, gpointer glibrary)
Search for cell reference gcell_ref in glibrary.
- static void `scan_cell_reference_dependencies` (gpointer gcell, gpointer glibrary)
Scans cell references inside cell This function searches all the references in gcell and updates the gds_cell_instance::cell_ref field in each instance.
- static void `scan_library_references` (gpointer glibrary_list_item, gpointer user)
Scans library's cell references.
- static void `gds_parse_date` (const char *buffer, int length, struct `gds_time_field` *mod_date, struct `gds_time_field` *access_date)
gds_parse_date
- int `parse_gds_from_file` (const char *filename, GList **library_list)
- static void `delete_cell_inst_element` (struct `gds_cell_instance` *cell_inst)
delete_cell_inst_element
- static void `delete_vertex` (struct `gds_point` *vertex)
delete_vertex
- static void `delete_graphics_obj` (struct `gds_graphics` *gfx)
delete_graphics_obj
- static void `delete_cell_element` (struct `gds_cell` *cell)
delete_cell_element
- static void `delete_library_element` (struct `gds_library` *lib)
delete_library_element
- int `clear_lib_list` (GList **library_list)
Deletes all libraries including cells, references etc.
- int `gds_tree_check_cell_references` (struct `gds_library` *lib)

- gds_tree_check_cell_references* checks if all child cell references can be resolved in the given library
- static int [gds_tree_check_list_contains_cell](#) (GList *list, struct [gds_cell](#) *cell)
Check if list contains a cell.
 - static int [gds_tree_check_iterate_ref_and_check](#) (struct [gds_cell](#) *cell_to_check, GList **visited_cells)
This function follows down the reference list of a cell and marks each visited subcell and detects loops.
 - int [gds_tree_check_reference_loops](#) (struct [gds_library](#) *lib)
gds_tree_check_reference_loops checks if the given library contains reference loops

11.11.1 Detailed Description

11.11.2 Macro Definition Documentation

11.11.2.1 CELL_NAME_MAX

```
#define CELL_NAME_MAX (100)
```

Maximum length of a [gds_cell::name](#) or a [gds_library::name](#).

Definition at line 37 of file [gds-types.h](#).

11.11.2.2 GDS_DEFAULT_UNITS

```
#define GDS_DEFAULT_UNITS (10E-9)
```

Default units assumed for library.

Note

This value is usually overwritten with the value defined in the library.

Definition at line 50 of file [gds-parser.c](#).

11.11.2.3 GDS_ERROR

```
#define GDS_ERROR(  
    fmt,  
    ... ) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)
```

Print GDS error.

Definition at line 52 of file [gds-parser.c](#).

11.11.2.4 GDS_INF

```
#define GDS_INF(  
    fmt,  
    ... )
```

Definition at line 58 of file [gds-parser.c](#).

11.11.2.5 GDS_PRINT_DEBUG_INFOS

```
#define GDS_PRINT_DEBUG_INFOS (0)
```

1: Print infos, 0: Don't print

Definition at line 38 of file [gds-parser.h](#).

11.11.2.6 GDS_WARN

```
#define GDS_WARN(  
    fmt,  
    ... ) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)
```

Print GDS warning.

Definition at line 53 of file [gds-parser.c](#).

11.11.2.7 MAX

```
#define MAX(  
    a,  
    b ) ((a) > (b)) ? (a) : (b)
```

Return bigger number.

Definition at line 41 of file [gds-types.h](#).

11.11.2.8 MIN

```
#define MIN(  
    a,  
    b ) ((a) < (b)) ? (a) : (b)
```

Return smaller number.

Definition at line 40 of file [gds-types.h](#).

11.11.3 Enumeration Type Documentation

11.11.3.1 anonymous enum

anonymous enum

Definition of check counter default value that indicates that the corresponding check has not yet been executed.

Enumerator

GDS_CELL_CHECK_NOT_RUN	
------------------------	--

Definition at line 45 of file [gds-types.h](#).

11.11.3.2 gds_record

enum [gds_record](#)

Enumerator

INVALID	
HEADER	
BGNLIB	
LIBNAME	
UNITS	
ENDLIB	
BGNSTR	
STRNAME	
ENDSTR	
BOUNDARY	
PATH	
SREF	
ENDEL	
XY	
MAG	
ANGLE	
SNAME	
STRANS	
BOX	
LAYER	
WIDTH	
PATHTYPE	

Definition at line 60 of file [gds-parser.c](#).

11.11.3.3 graphics_type

enum [graphics_type](#)

Types of graphic objects.

Enumerator

GRAPHIC_PATH	Path. Essentially a line.
--------------	---------------------------

Enumerator

GRAPHIC_POLYGON	An arbitrary polygon.
GRAPHIC_BOX	A rectangle. Warning Implementation in renderers might be buggy!

Definition at line 48 of file [gds-types.h](#).

11.11.3.4 path_type

enum [path_type](#)

Defines the line caps of a path.

Enumerator

PATH_FLUSH	
PATH_ROUNDED	
PATH_SQUARED	

Definition at line 58 of file [gds-types.h](#).

11.11.4 Function Documentation

11.11.4.1 append_cell()

```
static GList* append_cell (
    GList * curr_list,
    struct gds_cell ** cell_ptr ) [static]
```

`append_cell` Append a [gds_cell](#) to a list

Usage similar to [append_cell_ref\(\)](#).

Parameters

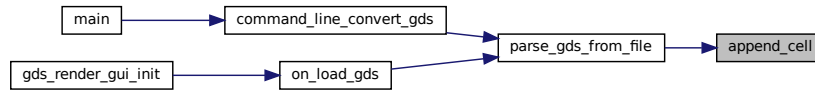
<i>curr_list</i>	List containing gds_cell elements. May be NULL
<i>cell_ptr</i>	newly created cell

Returns

new pointer to list

Definition at line 292 of file [gds-parser.c](#).

Here is the caller graph for this function:

**11.11.4.2 append_cell_ref()**

```

static GList* append_cell_ref (
    GList * curr_list,
    struct gds\_cell\_instance ** instance_ptr ) [static]
  
```

Append a cell reference to the reference GList.

Appends a new [gds_cell_instance](#) to `curr_list` and returns the new element via `instance_ptr`

Parameters

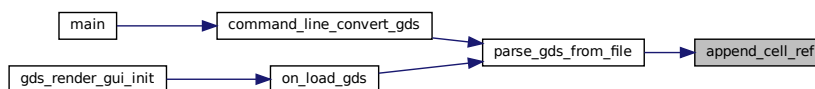
<i>curr_list</i>	List of gds_cell_instance elements. May be NULL
<i>instance_ptr</i>	newly created element

Returns

new GList pointer

Definition at line 321 of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.3 `append_graphics()`

```
static GList* append_graphics (
    GList * curr_list,
    enum graphics_type type,
    struct gds_graphics ** graphics_ptr ) [static]
```

Append graphics to list.

Parameters

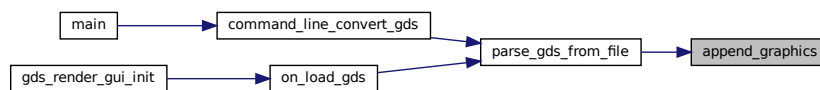
<i>curr_list</i>	List containing <code>gds_graphics</code> elements. May be NULL
<i>type</i>	Type of graphics
<i>graphics_ptr</i>	newly created graphic is written here

Returns

new list pointer

Definition at line 242 of file `gds-parser.c`.

Here is the caller graph for this function:



11.11.4.4 `append_library()`

```
static GList* append_library (
    GList * curr_list,
    struct gds_library ** library_ptr ) [static]
```

Append library to list.

Parameters

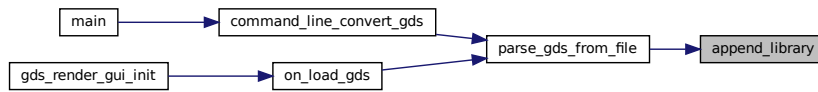
<i>curr_list</i>	List containing <code>gds_library</code> elements. May be NULL.
<i>library_ptr</i>	Return of newly created library.

Returns

Newly created list pointer

Definition at line 217 of file `gds-parser.c`.

Here is the caller graph for this function:



11.11.4.5 append_vertex()

```

static GList* append_vertex (
    GList * curr_list,
    int x,
    int y ) [static]
  
```

Appends vertex List.

Parameters

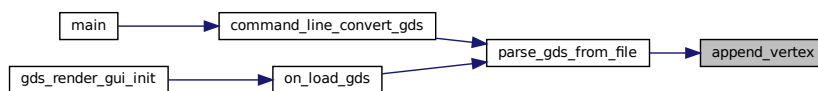
<i>curr_list</i>	List containing gds_point elements. May be NULL.
<i>x</i>	x-coordinate of new point
<i>y</i>	y-coordinate of new point

Returns

new Pointer to List.

Definition at line 271 of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.6 clear_lib_list()

```

int clear_lib_list (
    GList ** library_list )
  
```

Deletes all libraries including cells, references etc.

Parameters

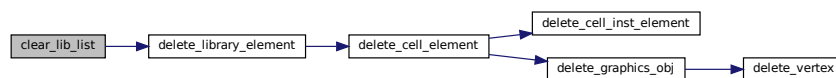
<i>library_list</i>	Pointer to a list of gds_library . Is set to NULL after completion.
---------------------	---

Returns

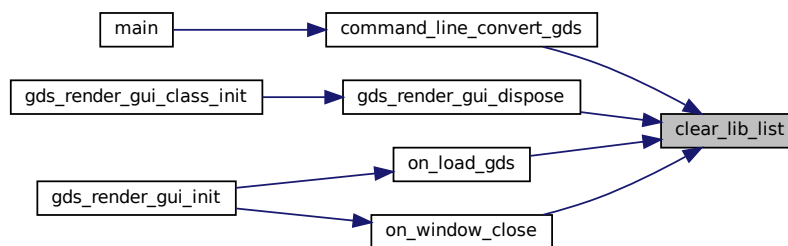
0

Definition at line [956](#) of file [gds-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.7 delete_cell_element()

```
static void delete_cell_element (
    struct gds\_cell * cell ) [static]
```

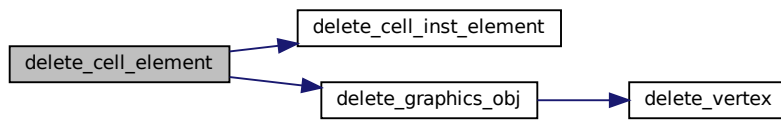
delete_cell_element

Parameters

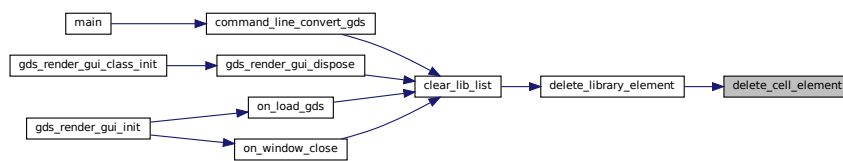
<i>cell</i>	
-------------	--

Definition at line [932](#) of file [gds-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.8 delete_cell_inst_element()

```
static void delete_cell_inst_element (
    struct gds_cell_instance * cell_inst ) [static]
```

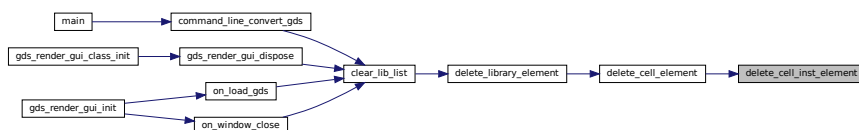
delete_cell_inst_element

Parameters

<i>cell_inst</i>	
------------------	--

Definition at line 899 of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.9 delete_graphics_obj()

```
static void delete_graphics_obj (
    struct gds_graphics * gfx ) [static]
```

delete_graphics_obj

Parameters

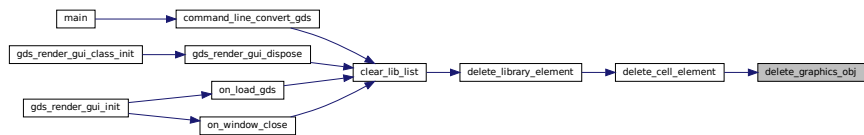
<i>gfx</i>	
------------	--

Definition at line 919 of file [gds-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.10 delete_library_element()

```
static void delete_library_element (
    struct gds_library * lib ) [static]
```

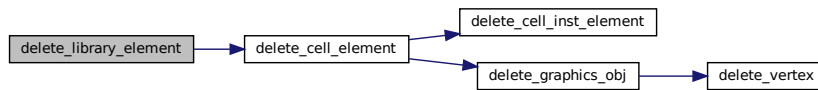
delete_library_element

Parameters

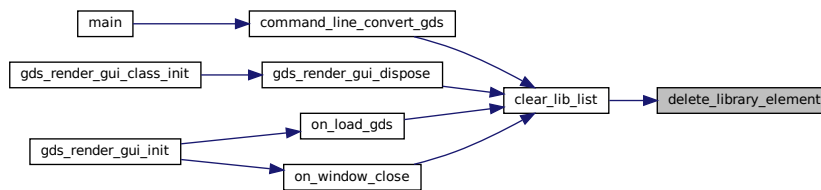
<i>lib</i>	
------------	--

Definition at line 946 of file [gds-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.11 delete_vertex()

```
static void delete_vertex (
    struct gds_point * vertex ) [static]
```

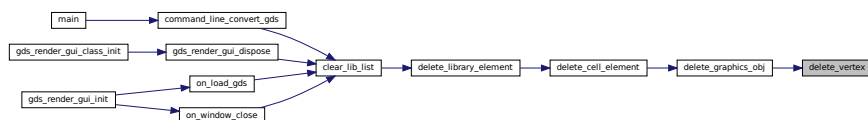
delete_vertex

Parameters

<i>vertex</i>	
---------------	--

Definition at line 909 of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.12 gds_convert_double()

```
static double gds_convert_double (
    const char * data ) [static]
```

Convert GDS 8-byte real to double.

Parameters

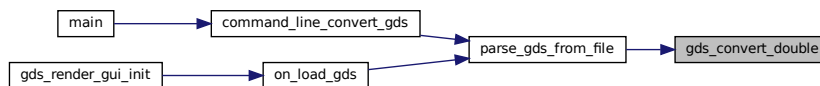
<i>data</i>	8 Byte GDS real
-------------	-----------------

Returns

result

Definition at line 120 of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.13 gds_convert_signed_int()

```
static signed int gds_convert_signed_int (
    const char * data ) [static]
```

Convert GDS INT32 to int.

Parameters

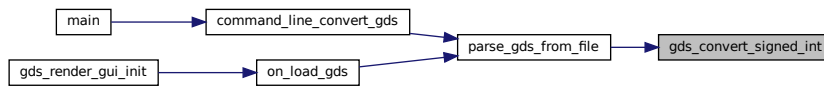
<i>data</i>	Buffer containing the int
-------------	---------------------------

Returns

result

Definition at line 165 of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.14 gds_convert_signed_int16()

```
static int16_t gds_convert_signed_int16 (
    const char * data ) [static]
```

Convert GDS INT16 to int16.

Parameters

<i>data</i>	Buffer containing the INT16
-------------	-----------------------------

Returns

result

Definition at line [186](#) of file [gds-parser.c](#).

Here is the caller graph for this function:



11.11.4.15 gds_convert_unsigend_int16()

```
static uint16_t gds_convert_unsigend_int16 (
    const char * data ) [static]
```

Convert GDS UINT16 String to uint16.

Parameters

<i>data</i>	Buffer containing the uint16
-------------	------------------------------

Returns

result

Definition at line 201 of file [gds-parser.c](#).

Here is the caller graph for this function:

**11.11.4.16 gds_parse_date()**

```

static void gds_parse_date (
    const char * buffer,
    int length,
    struct gds_time_field * mod_date,
    struct gds_time_field * access_date ) [static]
  
```

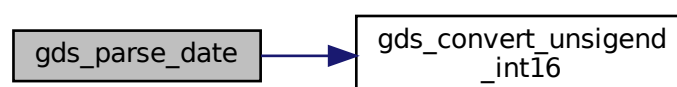
`gds_parse_date`

Parameters

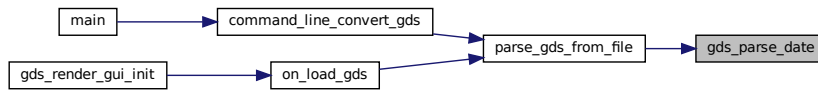
<i>buffer</i>	Buffer that contains the GDS Date field
<i>length</i>	Length of <i>buffer</i>
<i>mod_date</i>	Modification Date
<i>access_date</i>	Last Access Date

Definition at line 478 of file [gds-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.17 `gds_tree_check_cell_references()`

```
int gds_tree_check_cell_references (
    struct gds_library * lib )
```

`gds_tree_check_cell_references` checks if all child cell references can be resolved in the given library

This function will only mark cells that directly contain unresolved references.

If a cell contains a reference to a cell with unresolved references, it is not flagged.

Parameters

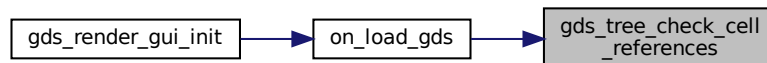
<code>lib</code>	The GDS library to check
------------------	--------------------------

Returns

less than 0 if an error occurred during processing; 0 if all child cells could be resolved; greater than zero if the processing was successful but not all cell references could be resolved. In this case the number of unresolved references is returned

Definition at line 40 of file [gds-tree-checker.c](#).

Here is the caller graph for this function:



11.11.4.18 `gds_tree_check_iterate_ref_and_check()`

```
static int gds_tree_check_iterate_ref_and_check (
    struct gds_cell * cell_to_check,
    GList ** visited_cells ) [static]
```

This function follows down the reference list of a cell and marks each visited subcell and detects loops.

Parameters

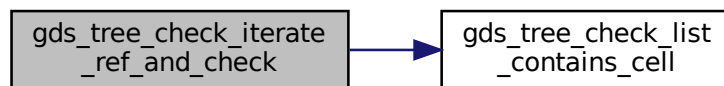
<i>cell_to_check</i>	The cell to check for reference loops
<i>visited_cells</i>	Pointer to list head. May be zero.

Returns

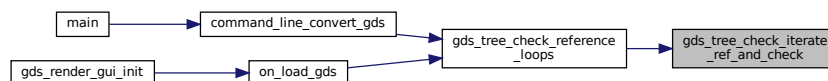
0 if no loops exist; error in processing: <0; loop found: >0

Definition at line 110 of file [gds-tree-checker.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.19 gds_tree_check_list_contains_cell()

```

static int gds_tree_check_list_contains_cell (
    GList * list,
    struct gds_cell * cell ) [static]
  
```

Check if list contains a cell.

Parameters

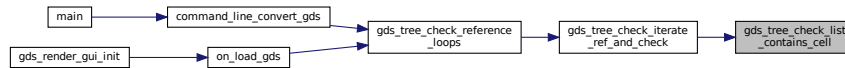
<i>list</i>	GList to check. May be a null pointer
<i>cell</i>	Cell to check for

Returns

0 if cell is not in list. 1 if cell is in list

Definition at line 93 of file [gds-tree-checker.c](#).

Here is the caller graph for this function:

**11.11.4.20 gds_tree_check_reference_loops()**

```
int gds_tree_check_reference_loops (
    struct gds_library * lib )
```

`gds_tree_check_reference_loops` checks if the given library contains reference loops

Parameters

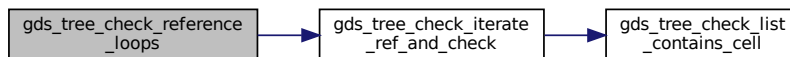
<i>lib</i>	GDS library
------------	-------------

Returns

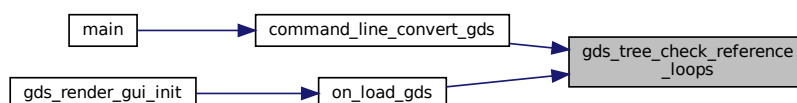
negative if an error occurred, zero if there are no reference loops, else a positive number representing the number of affected cells

Definition at line 157 of file [gds-tree-checker.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.21 name_cell()

```
static int name_cell (
    struct gds_cell * cell,
    unsigned int bytes,
    char * data,
    struct gds_library * lib ) [static]
```

Names a `gds_cell`.

Parameters

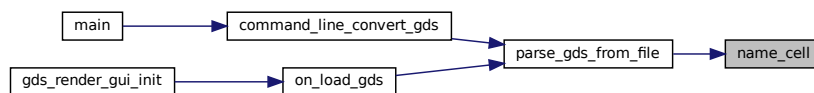
<i>cell</i>	Cell to name
<i>bytes</i>	Length of name
<i>data</i>	Name
<i>lib</i>	Library in which <code>cell</code> is located

Returns

0 id successful

Definition at line 380 of file `gds-parser.c`.

Here is the caller graph for this function:



11.11.4.22 name_cell_ref()

```
static int name_cell_ref (
    struct gds_cell_instance * cell_inst,
    unsigned int bytes,
    char * data ) [static]
```

Name cell reference.

Parameters

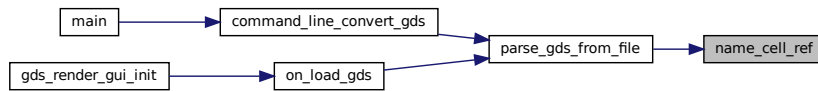
<i>cell_inst</i>	Cell reference
<i>bytes</i>	Length of name
<i>data</i>	Name

Returns

0 if successful

Definition at line 92 of file [gds-parser.c](#).

Here is the caller graph for this function:

**11.11.4.23 name_library()**

```

static int name_library (
    struct gds_library * current_library,
    unsigned int bytes,
    char * data ) [static]
  
```

Name a [gds_library](#).

Parameters

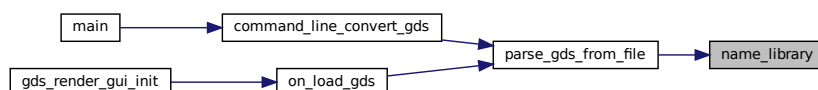
<i>current_library</i>	Library to name
<i>bytes</i>	Length of name
<i>data</i>	Name

Returns

0 if successful

Definition at line 349 of file [gds-parser.c](#).

Here is the caller graph for this function:

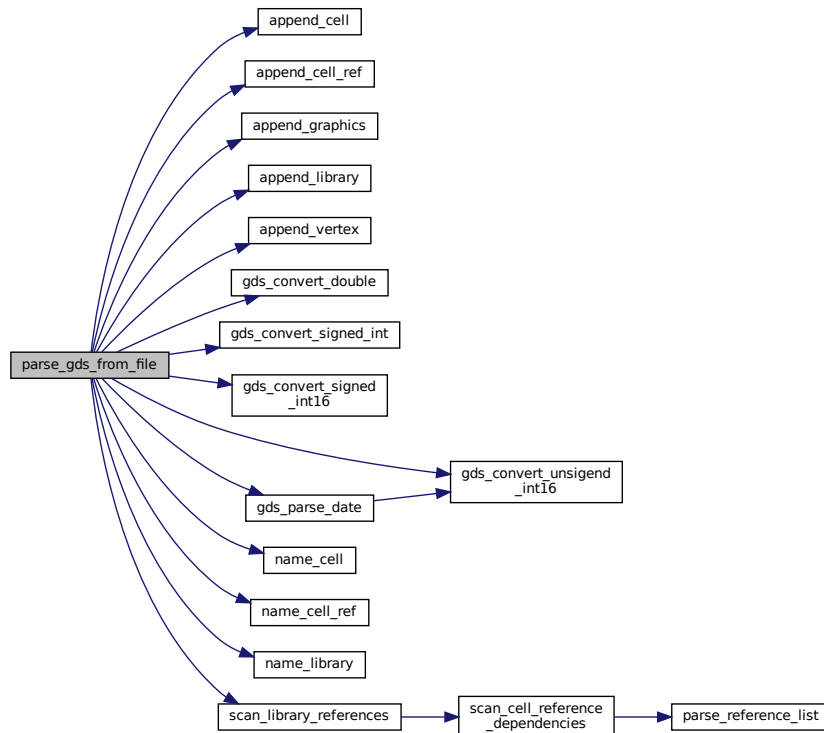


11.11.4.24 parse_gds_from_file()

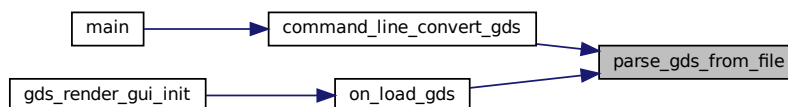
```
int parse_gds_from_file (
    const char * filename,
    GList ** library_list )
```

Definition at line 512 of file [gds-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.25 parse_reference_list()

```
static void parse_reference_list (
    gpointer gcell_ref,
    gpointer glibrary ) [static]
```

Search for cell reference `gcell_ref` in `glibrary`.

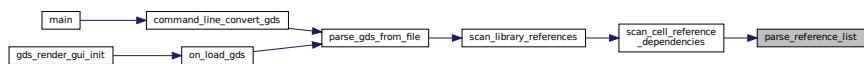
Search cell referenced by `gcell_ref` inside `glibrary` and update `gds_cell_instance::cell_ref` with found `gds_cell`

Parameters

<code>gcell_ref</code>	gpointer cast of struct <code>gds_cell_instance *</code>
<code>glibrary</code>	gpointer cast of struct <code>gds_library *</code>

Definition at line 412 of file `gds-parser.c`.

Here is the caller graph for this function:



11.11.4.26 scan_cell_reference_dependencies()

```
static void scan_cell_reference_dependencies (
    gpointer gcell,
    gpointer library ) [static]
```

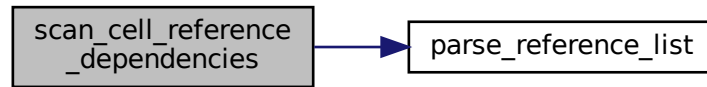
Scans cell references inside cell This function searches all the references in `gcell` and updates the `gds_cell_instance::cell_ref` field in each instance.

Parameters

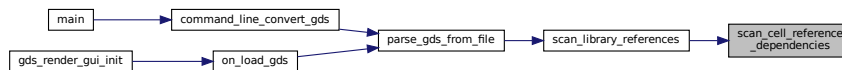
<code>gcell</code>	pointer cast of <code>gds_cell *</code>
<code>library</code>	Library where the cell references are searched in

Definition at line 444 of file `gds-parser.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



11.11.4.27 scan_library_references()

```

static void scan_library_references (
    gpointer library_list_item,
    gpointer user ) [static]
  
```

Scans library's cell references.

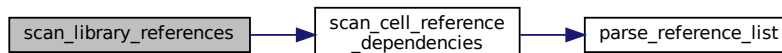
This function searches all the references between cells and updates the `gds_cell_instance::cell_ref` field in each instance

Parameters

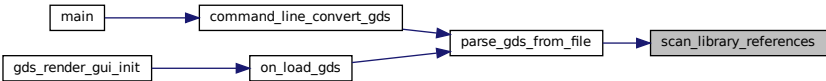
<i>library_list_item</i>	List containing <code>gds_library</code> elements
<i>user</i>	not used

Definition at line 462 of file `gds-parser.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



11.12 Mapping-Parser

Functions

- int [mapping_parser_load_line](#) (GDataInputStream *stream, gboolean *export, char **name, int *layer, GdkRGBA *color)
Load a line from stream and parse try to parse it as layer information.
- void [mapping_parser_gen_csv_line](#) (LayerElement *layer_element, char *line_buffer, size_t max_len)
Create Line for LayerMapping file with supplied information.

11.12.1 Detailed Description

11.12.2 Function Documentation

11.12.2.1 mapping_parser_gen_csv_line()

```
void mapping_parser_gen_csv_line (  
    LayerElement * layer_element,  
    char * line_buffer,  
    size_t max_len )
```

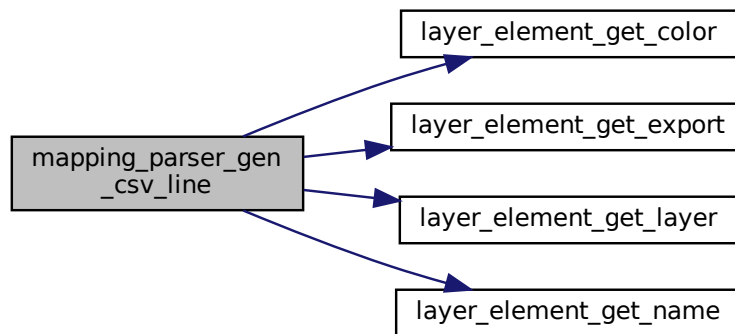
Create Line for LayerMapping file with supplied information.

Parameters

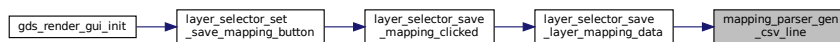
<i>layer_element</i>	information
<i>line_buffer</i>	buffer to write to
<i>max_len</i>	Maximum length that can be used in <i>line_buffer</i>

Definition at line 97 of file [mapping-parser.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.12.2.2 mapping_parser_load_line()

```

int mapping_parser_load_line (
    GDataInputStream * stream,
    gboolean * export,
    char ** name,
    int * layer,
    GdkRGBA * color )
  
```

Load a line from `stream` and parse try to parse it as layer information.

Parameters

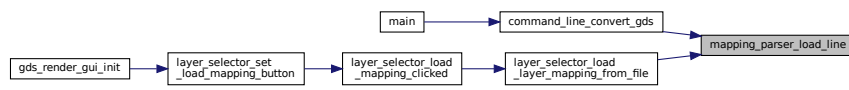
<i>stream</i>	Input data stream
<i>export</i>	Layer shall be exported
<i>name</i>	Layer name. Free returned pointer after using.
<i>layer</i>	Layer number
<i>color</i>	RGBA color.

Returns

1 if malformed line, 0 if parsing was successful and parameters are valid, -1 if file end

Definition at line 34 of file [mapping-parser.c](#).

Here is the caller graph for this function:



11.13 Version Number

See [Git Based Version Number](#).

Variables

- `const char * _app_version_string`
This string holds the [Git Based Version Number](#) of the app.
- `const char * _app_version_string = "! version not set !"`
This string holds the [Git Based Version Number](#) of the app.

11.13.1 Detailed Description

See [Git Based Version Number](#).

11.13.2 Variable Documentation

11.13.2.1 `_app_version_string` [1/2]

```
const char* _app_version_string
```

This string holds the [Git Based Version Number](#) of the app.

Definition at line 33 of file [version.c](#).

11.13.2.2 `_app_version_string` [2/2]

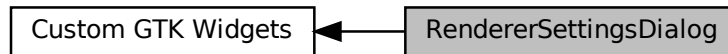
```
const char* _app_version_string = "! version not set !"
```

This string holds the [Git Based Version Number](#) of the app.

Definition at line 33 of file [version.c](#).

11.14 RendererSettingsDialog

Collaboration diagram for RendererSettingsDialog:



Data Structures

- struct [render_settings](#)
This struct holds the renderer configuration.
- struct [_RendererSettingsDialog](#)

Macros

- #define [RENDERER_TYPE_SETTINGS_DIALOG](#) ([renderer_settings_dialog_get_type\(\)](#))

Enumerations

- enum [output_renderer](#) { [RENDERER_LATEX_TIKZ](#), [RENDERER_CAIROGRAPHICS_PDF](#), [RENDERER_CAIROGRAPHICS_PNG](#) }
return type of the RedererSettingsDialog
- enum { [PROP_CELL_NAME](#) = 1, [PROP_COUNT](#) }

Functions

- [RendererSettingsDialog * renderer_settings_dialog_new](#) ([GtkWindow *parent](#))
Create a new RedererSettingsDialog GObject.
- [G_END_DECLS void renderer_settings_dialog_set_settings](#) ([RendererSettingsDialog *dialog](#), [struct render_settings *settings](#))
Apply settings to dialog.
- [void renderer_settings_dialog_get_settings](#) ([RendererSettingsDialog *dialog](#), [struct render_settings *settings](#))
Get the settings configured in the dialog.
- [void renderer_settings_dialog_set_cell_width](#) ([RendererSettingsDialog *dialog](#), [unsigned int width](#))
renderer_settings_dialog_set_cell_width Set width for rendered cell
- [void renderer_settings_dialog_set_cell_height](#) ([RendererSettingsDialog *dialog](#), [unsigned int height](#))
renderer_settings_dialog_set_cell_height Set height for rendered cell
- [void renderer_settings_dialog_set_database_unit_scale](#) ([RendererSettingsDialog *dialog](#), [double unit_in_meters](#))
renderer_settings_dialog_set_database_unit_scale Set database scale
- [static void renderer_settings_dialog_set_property](#) ([GObject *object](#), [guint property_id](#), [const GValue *value](#), [GParamSpec *pspec](#))

- static void [renderer_settings_dialog_get_property](#) (GObject *object, guint property_id, GValue *value, GParamSpec *pspec)
- static void [renderer_settings_dialog_class_init](#) (RendererSettingsDialogClass *klass)
- static void [show_tex_options](#) (RendererSettingsDialog *self)
- static void [hide_tex_options](#) (RendererSettingsDialog *self)
- static void [latex_render_callback](#) (GtkToggleButton *radio, RendererSettingsDialog *dialog)
- static gboolean [shape_drawer_drawing_callback](#) (GtkWidget *widget, cairo_t *cr, gpointer data)
- static double [convert_number_to_engineering](#) (double input, const char **out_prefix)
- static void [renderer_settings_dialog_update_labels](#) (RendererSettingsDialog *self)
- static void [scale_value_changed](#) (GtkRange *range, gpointer user_data)
- static void [renderer_settings_dialog_init](#) (RendererSettingsDialog *self)

Variables

- static GParamSpec * [properties](#) [PROP_COUNT]

11.14.1 Detailed Description

11.14.2 Macro Definition Documentation

11.14.2.1 RENDERER_TYPE_SETTINGS_DIALOG

```
#define RENDERER_TYPE_SETTINGS_DIALOG (renderer_settings_dialog_get_type())
```

Definition at line 51 of file [conv-settings-dialog.h](#).

11.14.3 Enumeration Type Documentation

11.14.3.1 anonymous enum

anonymous enum

Enumerator

PROP_CELL_NAME	
PROP_COUNT	

Definition at line 57 of file [conv-settings-dialog.c](#).

11.14.3.2 output_renderer

enum `output_renderer`

return type of the RedererSettingsDialog

Enumerator

RENDERER_LATEX_TIKZ	
RENDERER_CAIROGRAPHICS_PDF	
RENDERER_CAIROGRAPHICS_SVG	

Definition at line 40 of file `conv-settings-dialog.h`.

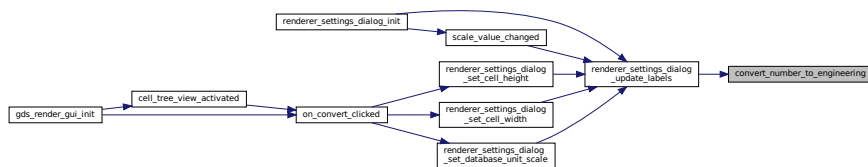
11.14.4 Function Documentation

11.14.4.1 convert_number_to_engineering()

```
static double convert_number_to_engineering (
    double input,
    const char ** out_prefix ) [static]
```

Definition at line 181 of file `conv-settings-dialog.c`.

Here is the caller graph for this function:

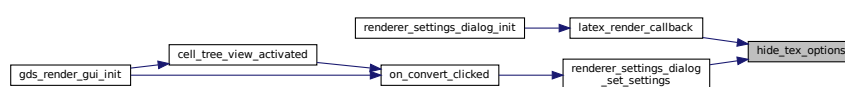


11.14.4.2 hide_tex_options()

```
static void hide_tex_options (
    RendererSettingsDialog * self ) [static]
```

Definition at line 120 of file `conv-settings-dialog.c`.

Here is the caller graph for this function:

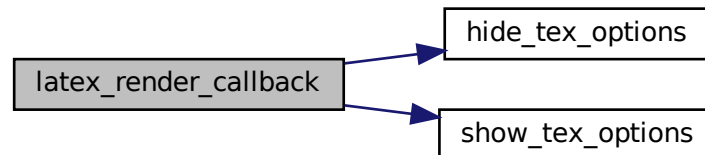


11.14.4.3 latex_render_callback()

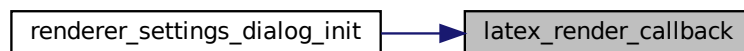
```
static void latex_render_callback (  
    GtkToggleButton * radio,  
    RendererSettingsDialog * dialog ) [static]
```

Definition at line 126 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

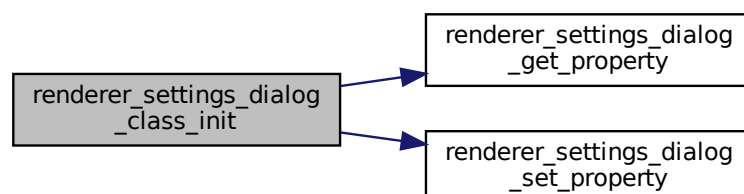


11.14.4.4 renderer_settings_dialog_class_init()

```
static void renderer_settings_dialog_class_init (  
    RendererSettingsDialogClass * klass ) [static]
```

Definition at line 97 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:

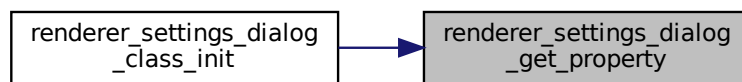


11.14.4.5 `renderer_settings_dialog_get_property()`

```
static void renderer_settings_dialog_get_property (
    GObject * object,
    guint property_id,
    GValue * value,
    GParamSpec * pspec ) [static]
```

Definition at line 81 of file [conv-settings-dialog.c](#).

Here is the caller graph for this function:



11.14.4.6 `renderer_settings_dialog_get_settings()`

```
void renderer_settings_dialog_get_settings (
    RendererSettingsDialog * dialog,
    struct render\_settings * settings )
```

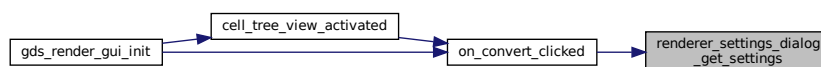
Get the settings configured in the dialog.

Parameters

<i>dialog</i>	
<i>settings</i>	

Definition at line 317 of file [conv-settings-dialog.c](#).

Here is the caller graph for this function:

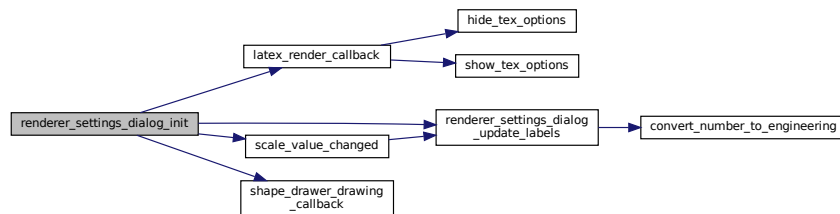


11.14.4.7 `renderer_settings_dialog_init()`

```
static void renderer_settings_dialog_init (
    RendererSettingsDialog * self ) [static]
```

Definition at line 265 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:

11.14.4.8 `renderer_settings_dialog_new()`

```
RendererSettingsDialog * renderer_settings_dialog_new (
    GtkWidget * parent )
```

Create a new RendererSettingsDialog GObject.

Parameters

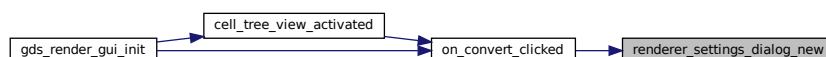
<i>parent</i>	Parent window
---------------	---------------

Returns

Created dialog object

Definition at line 306 of file [conv-settings-dialog.c](#).

Here is the caller graph for this function:



11.14.4.9 `renderer_settings_dialog_set_cell_height()`

```
void renderer_settings_dialog_set_cell_height (
    RendererSettingsDialog * dialog,
    unsigned int height )
```

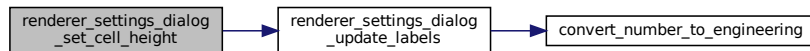
`renderer_settings_dialog_set_cell_height` Set height for rendered cell

Parameters

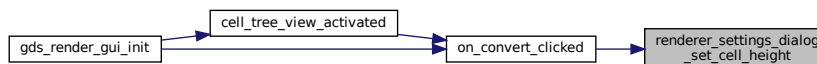
<i>dialog</i>	
<i>height</i>	Height in database units

Definition at line 374 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.14.4.10 `renderer_settings_dialog_set_cell_width()`

```
void renderer_settings_dialog_set_cell_width (
    RendererSettingsDialog * dialog,
    unsigned int width )
```

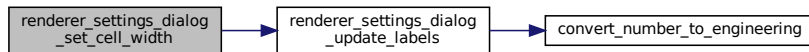
`renderer_settings_dialog_set_cell_width` Set width for rendered cell

Parameters

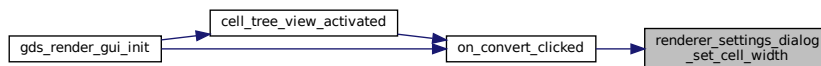
<i>dialog</i>	
<i>width</i>	Width in database units

Definition at line 362 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



11.14.4.11 renderer_settings_dialog_set_database_unit_scale()

```

void renderer_settings_dialog_set_database_unit_scale (
    RendererSettingsDialog * dialog,
    double unit_in_meters )
  
```

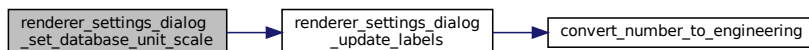
renderer_settings_dialog_set_database_unit_scale Set database scale

Parameters

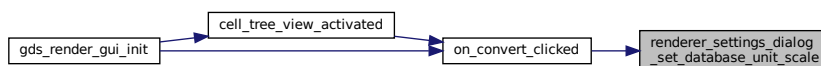
<i>dialog</i>	dialog element
<i>unit_in_meters</i>	Database unit in meters

Definition at line 386 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

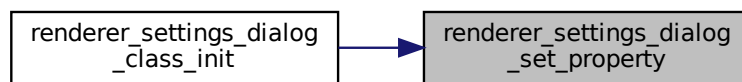


11.14.4.12 `renderer_settings_dialog_set_property()`

```
static void renderer_settings_dialog_set_property (
    GObject * object,
    guint property_id,
    const GValue * value,
    GParamSpec * pspec ) [static]
```

Definition at line 64 of file [conv-settings-dialog.c](#).

Here is the caller graph for this function:



11.14.4.13 `renderer_settings_dialog_set_settings()`

```
void renderer_settings_dialog_set_settings (
    RendererSettingsDialog * dialog,
    struct render\_settings * settings )
```

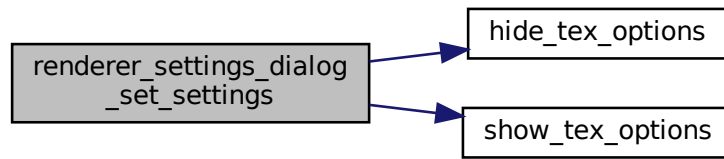
Apply settings to dialog.

Parameters

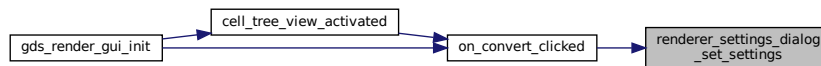
<i>dialog</i>	
<i>settings</i>	

Definition at line 337 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

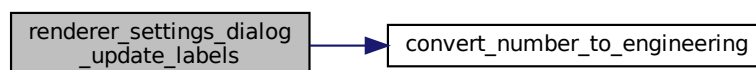


11.14.4.14 renderer_settings_dialog_update_labels()

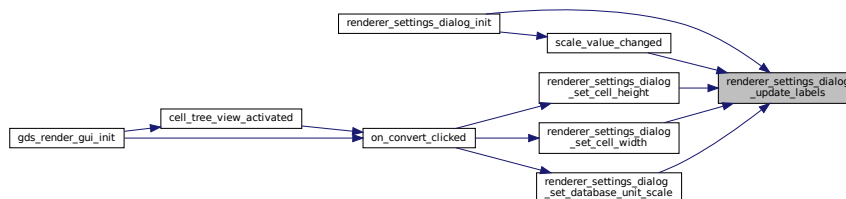
```
static void renderer_settings_dialog_update_labels (
    RendererSettingsDialog * self ) [static]
```

Definition at line 220 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

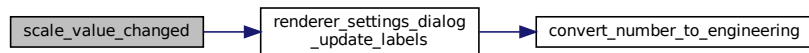


11.14.4.15 `scale_value_changed()`

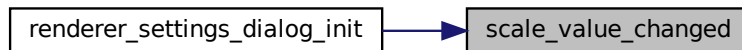
```
static void scale_value_changed (
    GtkRange * range,
    gpointer user_data ) [static]
```

Definition at line 256 of file [conv-settings-dialog.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

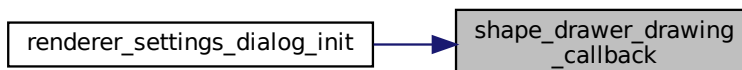


11.14.4.16 `shape_drawer_drawing_callback()`

```
static gboolean shape_drawer_drawing_callback (
    GtkWidget * widget,
    cairo_t * cr,
    gpointer data ) [static]
```

Definition at line 134 of file [conv-settings-dialog.c](#).

Here is the caller graph for this function:

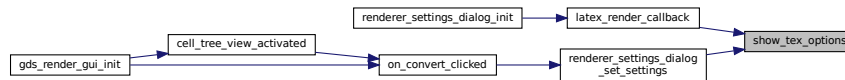


11.14.4.17 show_tex_options()

```
static void show_tex_options (  
    RendererSettingsDialog * self ) [static]
```

Definition at line 113 of file [conv-settings-dialog.c](#).

Here is the caller graph for this function:



11.14.5 Variable Documentation

11.14.5.1 properties

```
GParamSpec* properties[PROP_COUNT] [static]
```

Definition at line 62 of file [conv-settings-dialog.c](#).

11.15 LayerElement

Collaboration diagram for LayerElement:



Data Structures

- struct [_LayerElementPriv](#)
- struct [_LayerElement](#)
- struct [layer_element_dnd_data](#)

This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

Macros

- `#define TYPE_LAYER_ELEMENT (layer_element_get_type())`

Typedefs

- typedef struct [_LayerElementPriv](#) [LayerElementPriv](#)

Functions

- GtkWidget * [layer_element_new](#) (void)
Create new layer element object.
- const char * [layer_element_get_name](#) (LayerElement *elem)
get name of the layer
- void [layer_element_set_name](#) (LayerElement *elem, const char *name)
layer_element_set_name
- void [layer_element_set_layer](#) (LayerElement *elem, int layer)
Set layer number for this layer.
- int [layer_element_get_layer](#) (LayerElement *elem)
Get layer number.
- void [layer_element_set_export](#) (LayerElement *elem, gboolean export)
Set export flag for this layer.
- gboolean [layer_element_get_export](#) (LayerElement *elem)
Get export flag of layer.
- void [layer_element_get_color](#) (LayerElement *elem, GdkRGBA *rgba)
Get color of layer.
- void [layer_element_set_color](#) (LayerElement *elem, GdkRGBA *rgba)
Set color of layer.
- void [layer_element_set_dnd_callbacks](#) (LayerElement *elem, struct [layer_element_dnd_data](#) *data)
Setup drag and drop of elem for use in the LayerSelector.
- static void [layer_element_dispose](#) (GObject *obj)
- static void [layer_element_constructed](#) (GObject *obj)
- static void [layer_element_class_init](#) (LayerElementClass *klass)
- static void [layer_element_init](#) (LayerElement *self)

11.15.1 Detailed Description

11.15.2 Macro Definition Documentation

11.15.2.1 TYPE_LAYER_ELEMENT

```
#define TYPE_LAYER_ELEMENT (layer_element_get_type())
```

Definition at line 42 of file [layer-element.h](#).

11.15.3 Typedef Documentation

11.15.3.1 LayerElementPriv

```
typedef struct _LayerElementPriv LayerElementPriv
```

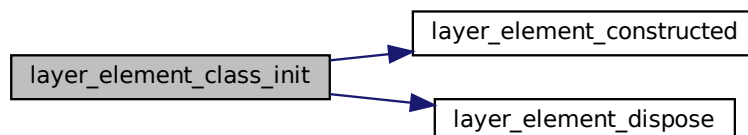
11.15.4 Function Documentation

11.15.4.1 layer_element_class_init()

```
static void layer_element_class_init (  
    LayerElementClass * klass ) [static]
```

Definition at line 54 of file [layer-element.c](#).

Here is the call graph for this function:



11.15.4.2 `layer_element_constructed()`

```
static void layer_element_constructed (
    GObject * obj ) [static]
```

Definition at line 49 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.3 `layer_element_dispose()`

```
static void layer_element_dispose (
    GObject * obj ) [static]
```

Definition at line 43 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.4 `layer_element_get_color()`

```
void layer_element_get_color (
    LayerElement * elem,
    GdkRGBA * rgba )
```

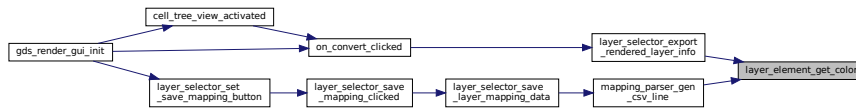
Get color of layer.

Parameters

<i>elem</i>	Layer Element
<i>rgba</i>	RGBA color

Definition at line 120 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.5 layer_element_get_export()

```
gboolean layer_element_get_export (
    LayerElement * elem )
```

Get export flag of layer.

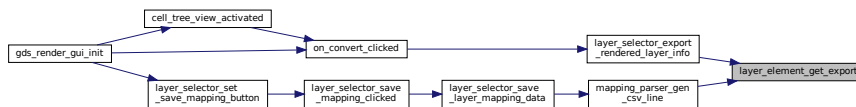
Parameters

<i>elem</i>	Layer Element
-------------	---------------

Returns

Definition at line 115 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.6 layer_element_get_layer()

```
int layer_element_get_layer (
    LayerElement * elem )
```

Get layer number.

Parameters

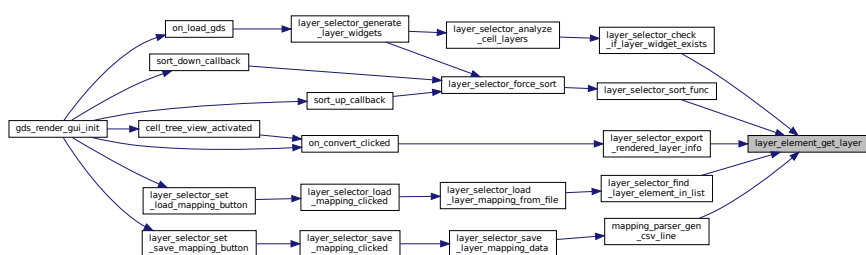
<i>elem</i>	Layer Element
-------------	---------------

Returns

Number of this layer

Definition at line 105 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.7 layer_element_get_name()

```
const char * layer_element_get_name (
    LayerElement * elem )
```

get name of the layer

Parameters

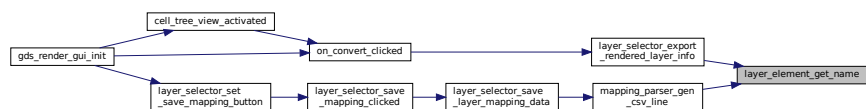
<i>elem</i>	Layer element
-------------	---------------

Returns

Name. Must not be changed, freed or anything else.

Definition at line 84 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.8 layer_element_init()

```
static void layer_element_init (
    LayerElement * self ) [static]
```

Definition at line 61 of file [layer-element.c](#).

11.15.4.9 layer_element_new()

```
GtkWidget * layer_element_new (
    void )
```

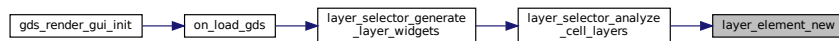
Create new layer element object.

Returns

new object

Definition at line 79 of file [layer-element.c](#).

Here is the caller graph for this function:

**11.15.4.10 layer_element_set_color()**

```
void layer_element_set_color (
    LayerElement * elem,
    GdkRGBA * rgba )
```

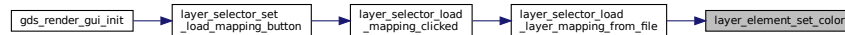
Set color of layer.

Parameters

<i>elem</i>	Layer Element
<i>rgba</i>	RGBA color

Definition at line 128 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.11 layer_element_set_dnd_callbacks()

```

void layer_element_set_dnd_callbacks (
    LayerElement * elem,
    struct layer_element_dnd_data * data )
  
```

Setup drag and drop of `elem` for use in the LayerSelector.

Parameters

<i>elem</i>	Layer element to set up
<i>data</i>	Data array containing the necessary callbacks etc. for drag and drop.

Definition at line [136](#) of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.12 layer_element_set_export()

```

void layer_element_set_export (
    LayerElement * elem,
    gboolean export )
  
```

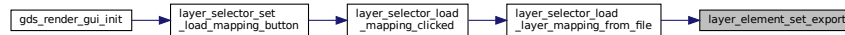
Set export flag for this layer.

Parameters

<i>elem</i>	Layer Element
<i>export</i>	flag

Definition at line [110](#) of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.13 layer_element_set_layer()

```

void layer_element_set_layer (
    LayerElement * elem,
    int layer )
  
```

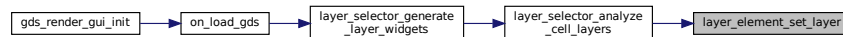
Set layer number for this layer.

Parameters

<i>elem</i>	Layer element
<i>layer</i>	Layer number

Definition at line 94 of file [layer-element.c](#).

Here is the caller graph for this function:



11.15.4.14 layer_element_set_name()

```

void layer_element_set_name (
    LayerElement * elem,
    const char * name )
  
```

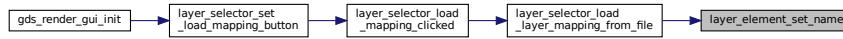
layer_element_set_name

Parameters

<i>elem</i>	set the name of the layer
<i>name</i>	Name. Can be freed after call to this function

Definition at line 89 of file [layer-element.c](#).

Here is the caller graph for this function:



Chapter 12

Data Structure Documentation

12.1 `gds_cell_checks::_check_internals` Struct Reference

For the internal use of the checker.

```
#include <gds-types.h>
```

Data Fields

- int [marker](#)

12.1.1 Detailed Description

For the internal use of the checker.

Warning

Do not use this structure and its contents!

Definition at line [78](#) of file [gds-types.h](#).

12.1.2 Field Documentation

12.1.2.1 `marker`

```
int marker
```

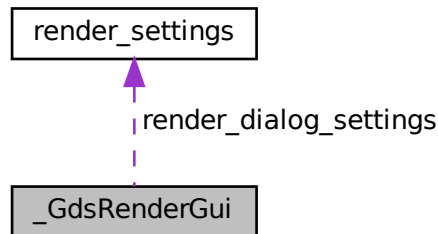
Definition at line [79](#) of file [gds-types.h](#).

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

- [gds-types.h](#)

12.2 `_GdsRenderGui` Struct Reference

Collaboration diagram for `_GdsRenderGui`:



Data Fields

- GObject [parent](#)
- GtkWidget * [main_window](#)
- GtkWidget * [convert_button](#)
- GtkTreeStore * [cell_tree_store](#)
- GtkWidget * [cell_search_entry](#)
- LayerSelector * [layer_selector](#)
- GtkTreeView * [cell_tree_view](#)
- GList * [gds_libraries](#)
- struct [render_settings](#) [render_dialog_settings](#)

12.2.1 Detailed Description

Definition at line 49 of file [gds-render-gui.c](#).

12.2.2 Field Documentation

12.2.2.1 `cell_search_entry`

GtkWidget* `cell_search_entry`

Definition at line 57 of file [gds-render-gui.c](#).

12.2.2.2 cell_tree_store

GtkTreeStore* cell_tree_store

Definition at line 56 of file [gds-render-gui.c](#).

12.2.2.3 cell_tree_view

GtkTreeView* cell_tree_view

Definition at line 59 of file [gds-render-gui.c](#).

12.2.2.4 convert_button

GtkWidget* convert_button

Definition at line 55 of file [gds-render-gui.c](#).

12.2.2.5 gds_libraries

GList* gds_libraries

Definition at line 60 of file [gds-render-gui.c](#).

12.2.2.6 layer_selector

LayerSelector* layer_selector

Definition at line 58 of file [gds-render-gui.c](#).

12.2.2.7 main_window

GtkWindow* main_window

Definition at line 54 of file [gds-render-gui.c](#).

12.2.2.8 parent

GObject parent

Definition at line 51 of file [gds-render-gui.c](#).

12.2.2.9 render_dialog_settings

```
struct render\_settings render_dialog_settings
```

Definition at line 61 of file [gds-render-gui.c](#).

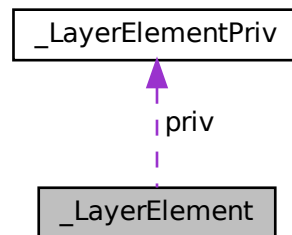
The documentation for this struct was generated from the following file:

- [gds-render-gui.c](#)

12.3 _LayerElement Struct Reference

```
#include <layer-element.h>
```

Collaboration diagram for _LayerElement:



Data Fields

- GtkListBoxRow [parent](#)
- [LayerElementPriv](#) `priv`

12.3.1 Detailed Description

Definition at line 53 of file [layer-element.h](#).

12.3.2 Field Documentation

12.3.2.1 `parent`

`GtkListBoxRow` `parent`

Definition at line 55 of file [layer-element.h](#).

12.3.2.2 `priv`

`LayerElementPriv` `priv`

Definition at line 57 of file [layer-element.h](#).

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

- [layer-element.h](#)

12.4 `_LayerElementPriv` Struct Reference

```
#include <layer-element.h>
```

Data Fields

- `GtkEntry` * [name](#)
- `GtkLabel` * [layer](#)
- `int` [layer_num](#)
- `GtkEventBox` * [event_handle](#)
- `GtkColorButton` * [color](#)
- `GtkCheckButton` * [export](#)

12.4.1 Detailed Description

Definition at line 44 of file [layer-element.h](#).

12.4.2 Field Documentation

12.4.2.1 color

`GtkColorButton* color`

Definition at line 49 of file [layer-element.h](#).

12.4.2.2 event_handle

`GtkEventBox* event_handle`

Definition at line 48 of file [layer-element.h](#).

12.4.2.3 export

`GtkCheckButton* export`

Definition at line 50 of file [layer-element.h](#).

12.4.2.4 layer

`GtkLabel* layer`

Definition at line 46 of file [layer-element.h](#).

12.4.2.5 layer_num

`int layer_num`

Definition at line 47 of file [layer-element.h](#).

12.4.2.6 name

`GtkEntry* name`

Definition at line 45 of file [layer-element.h](#).

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

- [layer-element.h](#)

12.5 `_LayerSelector` Struct Reference

Data Fields

- GObject `parent`
- GtkWidget * `associated_load_button`
- GtkWidget * `associated_save_button`
- GtkWindow * `load_parent_window`
- GtkWindow * `save_parent_window`
- GtkListBox * `list_box`
- GtkTargetEntry `dnd_target`
- gpointer `dummy` [4]

12.5.1 Detailed Description

Definition at line 42 of file `layer-selector.c`.

12.5.2 Field Documentation

12.5.2.1 `associated_load_button`

```
GtkWidget* associated_load_button
```

Definition at line 46 of file `layer-selector.c`.

12.5.2.2 `associated_save_button`

```
GtkWidget* associated_save_button
```

Definition at line 47 of file `layer-selector.c`.

12.5.2.3 `dnd_target`

```
GtkTargetEntry dnd_target
```

Definition at line 52 of file `layer-selector.c`.

12.5.2.4 dummy

```
gpointer dummy[4]
```

Definition at line 54 of file [layer-selector.c](#).

12.5.2.5 list_box

```
GtkListBox* list_box
```

Definition at line 50 of file [layer-selector.c](#).

12.5.2.6 load_parent_window

```
GtkWindow* load_parent_window
```

Definition at line 48 of file [layer-selector.c](#).

12.5.2.7 parent

```
GObject parent
```

Definition at line 44 of file [layer-selector.c](#).

12.5.2.8 save_parent_window

```
GtkWindow* save_parent_window
```

Definition at line 49 of file [layer-selector.c](#).

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

- [layer-selector.c](#)

12.6 `_LibCellRenderer` Struct Reference

```
#include <lib-cell-renderer.h>
```


Data Fields

- `GtkCellRendererText` [super](#)

12.6.1 Detailed Description

Definition at line 48 of file `lib-cell-renderer.h`.

12.6.2 Field Documentation

12.6.2.1 `super`

`GtkCellRendererText` `super`

Definition at line 50 of file `lib-cell-renderer.h`.

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

- `lib-cell-renderer.h`

12.7 `_RendererSettingsDialog` Struct Reference

Data Fields

- `GtkDialog` [parent](#)
- `GtkWidget` * [radio_latex](#)
- `GtkWidget` * [radio_cairo_pdf](#)
- `GtkWidget` * [radio_cairo_svg](#)
- `GtkWidget` * [scale](#)
- `GtkWidget` * [layer_check](#)
- `GtkWidget` * [standalone_check](#)
- `GtkDrawingArea` * [shape_drawing](#)
- `GtkLabel` * [x_label](#)
- `GtkLabel` * [y_label](#)
- `GtkLabel` * [x_output_label](#)
- `GtkLabel` * [y_output_label](#)
- unsigned int [cell_height](#)
- unsigned int [cell_width](#)
- double [unit_in_meters](#)

12.7.1 Detailed Description

Definition at line 34 of file `conv-settings-dialog.c`.

12.7.2 Field Documentation

12.7.2.1 `cell_height`

`unsigned int cell_height`

Definition at line 50 of file [conv-settings-dialog.c](#).

12.7.2.2 `cell_width`

`unsigned int cell_width`

Definition at line 51 of file [conv-settings-dialog.c](#).

12.7.2.3 `layer_check`

`GtkWidget* layer_check`

Definition at line 41 of file [conv-settings-dialog.c](#).

12.7.2.4 `parent`

`GtkDialog parent`

Definition at line 35 of file [conv-settings-dialog.c](#).

12.7.2.5 `radio_cairo_pdf`

`GtkWidget* radio_cairo_pdf`

Definition at line 38 of file [conv-settings-dialog.c](#).

12.7.2.6 radio_cairo_svg

GtkWidget* radio_cairo_svg

Definition at line 39 of file [conv-settings-dialog.c](#).

12.7.2.7 radio_latex

GtkWidget* radio_latex

Definition at line 37 of file [conv-settings-dialog.c](#).

12.7.2.8 scale

GtkWidget* scale

Definition at line 40 of file [conv-settings-dialog.c](#).

12.7.2.9 shape_drawing

GtkDrawingArea* shape_drawing

Definition at line 43 of file [conv-settings-dialog.c](#).

12.7.2.10 standalone_check

GtkWidget* standalone_check

Definition at line 42 of file [conv-settings-dialog.c](#).

12.7.2.11 unit_in_meters

double unit_in_meters

Definition at line 52 of file [conv-settings-dialog.c](#).

12.7.2.12 x_label

GtkLabel* x_label

Definition at line 44 of file [conv-settings-dialog.c](#).

12.7.2.13 x_output_label

GtkLabel* x_output_label

Definition at line 47 of file [conv-settings-dialog.c](#).

12.7.2.14 y_label

GtkLabel* y_label

Definition at line 45 of file [conv-settings-dialog.c](#).

12.7.2.15 y_output_label

GtkLabel* y_output_label

Definition at line 48 of file [conv-settings-dialog.c](#).

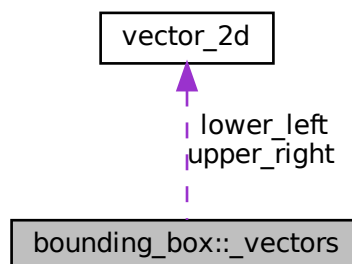
The documentation for this struct was generated from the following file:

- [conv-settings-dialog.c](#)

12.8 bounding_box::_vectors Struct Reference

```
#include <bounding-box.h>
```

Collaboration diagram for bounding_box::_vectors:



Data Fields

- struct [vector_2d](#) [lower_left](#)
- struct [vector_2d](#) [upper_right](#)

12.8.1 Detailed Description

Coordinate System is (y up | x right)

Definition at line 40 of file [bounding-box.h](#).

12.8.2 Field Documentation

12.8.2.1 lower_left

```
struct vector\_2d lower_left
```

Definition at line 41 of file [bounding-box.h](#).

12.8.2.2 upper_right

```
struct vector\_2d upper_right
```

Definition at line 42 of file [bounding-box.h](#).

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

- [bounding-box.h](#)

12.9 application_data Struct Reference

Structure containing The GtkApplication and a list containing the GdsRenderGui objects.

Data Fields

- GtkApplication * [app](#)
- GList * [gui_list](#)

12.9.1 Detailed Description

Structure containing The GtkApplication and a list containing the GdsRenderGui objects.

Definition at line 38 of file [main.c](#).

12.9.2 Field Documentation

12.9.2.1 app

```
GtkApplication* app
```

Definition at line 39 of file [main.c](#).

12.9.2.2 gui_list

```
GList* gui_list
```

Definition at line 40 of file [main.c](#).

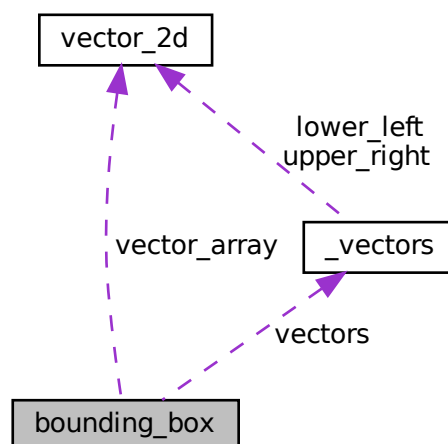
The documentation for this struct was generated from the following file:

- [main.c](#)

12.10 bounding_box Union Reference

```
#include <bounding-box.h>
```

Collaboration diagram for bounding_box:



Data Structures

- [struct _vectors](#)

Data Fields

- struct [bounding_box::_vectors](#) [vectors](#)
- struct [vector_2d](#) [vector_array](#) [2]

12.10.1 Detailed Description

Definition at line 38 of file [bounding-box.h](#).

12.10.2 Field Documentation

12.10.2.1 vector_array

```
struct vector\_2d vector\_array[2]
```

Definition at line 44 of file [bounding-box.h](#).

12.10.2.2 vectors

```
struct bounding\_box::\_vectors vectors
```

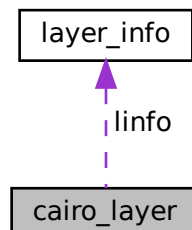
The documentation for this union was generated from the following file:

- [bounding-box.h](#)

12.11 cairo_layer Struct Reference

The [cairo_layer](#) struct Each rendered layer is represented by this struct.

Collaboration diagram for [cairo_layer](#):



Data Fields

- `cairo_t * cr`
cairo context for layer
- `cairo_surface_t * rec`
Recording surface to hold the layer.
- struct `layer_info * linfo`
Reference to layer information.

12.11.1 Detailed Description

The `cairo_layer` struct Each rendered layer is represented by this struct.

Definition at line 41 of file `cairo-output.c`.

12.11.2 Field Documentation

12.11.2.1 `cr`

```
cairo_t* cr
```

cairo context for layer

Definition at line 42 of file `cairo-output.c`.

12.11.2.2 `linfo`

```
struct layer_info* linfo
```

Reference to layer information.

Definition at line 44 of file `cairo-output.c`.

12.11.2.3 `rec`

```
cairo_surface_t* rec
```

Recording surface to hold the layer.

Definition at line 43 of file `cairo-output.c`.

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

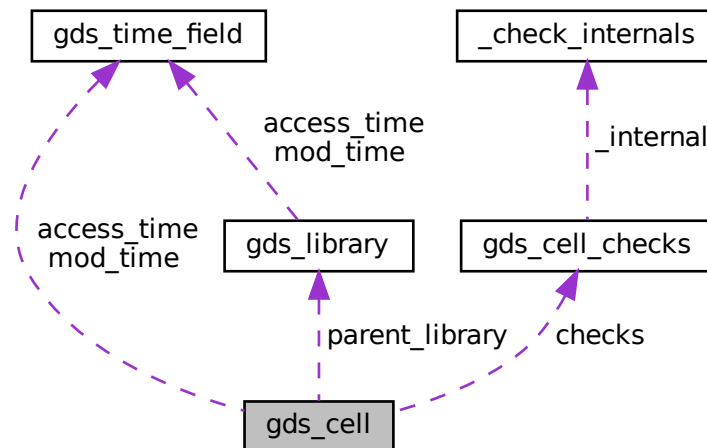
- `cairo-output.c`

12.12 gds_cell Struct Reference

A Cell inside a [gds_library](#).

```
#include <gds-types.h>
```

Collaboration diagram for gds_cell:



Data Fields

- char [name](#) [[CELL_NAME_MAX](#)]
- struct [gds_time_field](#) `mod_time`
- struct [gds_time_field](#) `access_time`
- GList * `child_cells`
List of [gds_cell_instance](#) elements.
- GList * `graphic_objs`
List of [gds_graphics](#).
- struct [gds_library](#) * `parent_library`
Pointer to parent library.
- struct [gds_cell_checks](#) `checks`
Checking results.

12.12.1 Detailed Description

A Cell inside a [gds_library](#).

Definition at line 122 of file [gds-types.h](#).

12.12.2 Field Documentation

12.12.2.1 access_time

```
struct gds_time_field access_time
```

Definition at line 125 of file [gds-types.h](#).

12.12.2.2 checks

```
struct gds_cell_checks checks
```

Checking results.

Definition at line 129 of file [gds-types.h](#).

12.12.2.3 child_cells

```
GList* child_cells
```

List of [gds_cell_instance](#) elements.

Definition at line 126 of file [gds-types.h](#).

12.12.2.4 graphic_objs

```
GList* graphic_objs
```

List of [gds_graphics](#).

Definition at line 127 of file [gds-types.h](#).

12.12.2.5 mod_time

```
struct gds_time_field mod_time
```

Definition at line 124 of file [gds-types.h](#).

12.12.2.6 name

```
char name[CELL_NAME_MAX]
```

Definition at line 123 of file [gds-types.h](#).

12.12.2.7 parent_library

```
struct gds_library* parent_library
```

Pointer to parent library.

Definition at line 128 of file [gds-types.h](#).

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

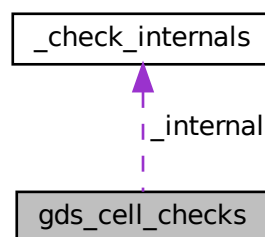
- [gds-types.h](#)

12.13 gds_cell_checks Struct Reference

Stores the result of the cell checks.

```
#include <gds-types.h>
```

Collaboration diagram for gds_cell_checks:



Data Structures

- [struct _check_internals](#)

For the internal use of the checker.

Data Fields

- int [unresolved_child_count](#)
Number of unresolved cell instances inside this cell. Default: [GDS_CELL_CHECK_NOT_RUN](#).
- int [affected_by_reference_loop](#)
1 if the cell is affected by a reference loop and therefore not renderable. Default: [GDS_CELL_CHECK_NOT_RUN](#)
- struct [gds_cell_checks::_check_internals _internal](#)

12.13.1 Detailed Description

Stores the result of the cell checks.

Definition at line 71 of file [gds-types.h](#).

12.13.2 Field Documentation

12.13.2.1 [_internal](#)

```
struct gds\_cell\_checks::\_check\_internals \_internal
```

12.13.2.2 [affected_by_reference_loop](#)

```
int affected\_by\_reference\_loop
```

1 if the cell is affected by a reference loop and therefore not renderable. Default: [GDS_CELL_CHECK_NOT_RUN](#)

Definition at line 73 of file [gds-types.h](#).

12.13.2.3 [unresolved_child_count](#)

```
int unresolved\_child\_count
```

Number of unresolved cell instances inside this cell. Default: [GDS_CELL_CHECK_NOT_RUN](#).

Definition at line 72 of file [gds-types.h](#).

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

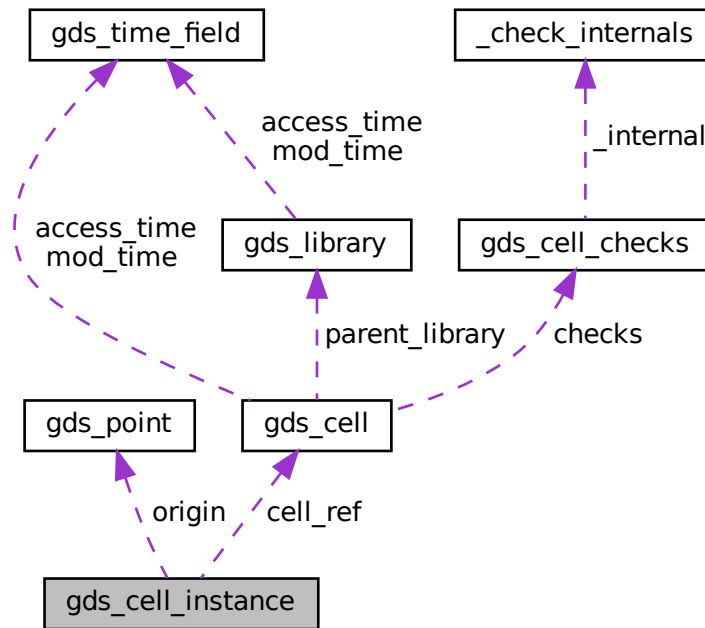
- [gds-types.h](#)

12.14 gds_cell_instance Struct Reference

This represents an instance of a cell inside another cell.

```
#include <gds-types.h>
```

Collaboration diagram for gds_cell_instance:



Data Fields

- char `ref_name` [`CELL_NAME_MAX`]
Name of referenced cell.
- struct `gds_cell * cell_ref`
Referenced `gds_cell` structure.
- struct `gds_point origin`
Origin.
- int `flipped`
Mirrored on x-axis before rotation.
- double `angle`
Angle of rotation (counter clockwise) in degrees.
- double `magnification`
magnification

12.14.1 Detailed Description

This represents an instance of a cell inside another cell.

Definition at line 110 of file `gds-types.h`.

12.14.2 Field Documentation

12.14.2.1 angle

```
double angle
```

Angle of rotation (counter clockwise) in degrees.

Definition at line 115 of file [gds-types.h](#).

12.14.2.2 cell_ref

```
struct gds\_cell* cell_ref
```

Referenced [gds_cell](#) structure.

Definition at line 112 of file [gds-types.h](#).

12.14.2.3 flipped

```
int flipped
```

Mirrored on x-axis before rotation.

Definition at line 114 of file [gds-types.h](#).

12.14.2.4 magnification

```
double magnification
```

magnification

Definition at line 116 of file [gds-types.h](#).

12.14.2.5 origin

```
struct gds\_point origin
```

Origin.

Definition at line 113 of file [gds-types.h](#).

12.14.2.6 ref_name

```
char ref_name[CELL_NAME_MAX]
```

Name of referenced cell.

Definition at line 111 of file [gds-types.h](#).

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

- [gds-types.h](#)

12.15 gds_graphics Struct Reference

A GDS graphics object.

```
#include <gds-types.h>
```

Data Fields

- enum [graphics_type](#) [gfx_type](#)
Type of graphic.
- GList * [vertices](#)
List of [gds_point](#).
- enum [path_type](#) [path_render_type](#)
Line cap.
- int [width_absolute](#)
Width. Not used for objects other than paths.
- int16_t [layer](#)
Layer the graphic object is on.
- uint16_t [datatype](#)

12.15.1 Detailed Description

A GDS graphics object.

Definition at line 98 of file [gds-types.h](#).

12.15.2 Field Documentation

12.15.2.1 datatype

```
uint16_t datatype
```

Definition at line 104 of file [gds-types.h](#).

12.15.2.2 gfx_type

```
enum graphics_type gfx_type
```

Type of graphic.

Definition at line 99 of file [gds-types.h](#).

12.15.2.3 layer

```
int16_t layer
```

Layer the graphic object is on.

Definition at line 103 of file [gds-types.h](#).

12.15.2.4 path_render_type

```
enum path_type path_render_type
```

Line cap.

Definition at line 101 of file [gds-types.h](#).

12.15.2.5 vertices

```
GList* vertices
```

List of [gds_point](#).

Definition at line 100 of file [gds-types.h](#).

12.15.2.6 width_absolute

```
int width_absolute
```

Width. Not used for objects other than paths.

Definition at line 102 of file [gds-types.h](#).

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

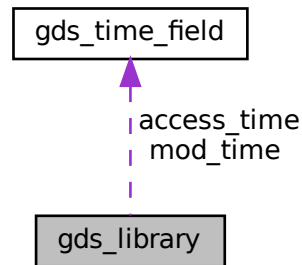
- [gds-types.h](#)

12.16 gds_library Struct Reference

GDS Toplevel library.

```
#include <gds-types.h>
```

Collaboration diagram for gds_library:



Data Fields

- char [name](#) [[CELL_NAME_MAX](#)]
- struct [gds_time_field](#) [mod_time](#)
- struct [gds_time_field](#) [access_time](#)
- double [unit_in_meters](#)
- GList * [cells](#)
- GList * [cell_names](#)

12.16.1 Detailed Description

GDS Toplevel library.

Definition at line [135](#) of file [gds-types.h](#).

12.16.2 Field Documentation

12.16.2.1 access_time

```
struct gds\_time\_field access\_time
```

Definition at line [138](#) of file [gds-types.h](#).

12.16.2.2 cell_names

```
GList* cell_names
```

< List of strings that contains all cell names

Definition at line 141 of file [gds-types.h](#).

12.16.2.3 cells

```
GList* cells
```

List of [gds_cell](#) that contains all cells in this library

Definition at line 140 of file [gds-types.h](#).

12.16.2.4 mod_time

```
struct gds_time_field mod_time
```

Definition at line 137 of file [gds-types.h](#).

12.16.2.5 name

```
char name[CELL_NAME_MAX]
```

Definition at line 136 of file [gds-types.h](#).

12.16.2.6 unit_in_meters

```
double unit_in_meters
```

Length of a database unit in meters

Definition at line 139 of file [gds-types.h](#).

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

- [gds-types.h](#)

12.17 gds_point Struct Reference

A point in the 2D plane. Sometimes references as vertex.

```
#include <gds-types.h>
```

Data Fields

- [int x](#)
- [int y](#)

12.17.1 Detailed Description

A point in the 2D plane. Sometimes references as vertex.

Definition at line [63](#) of file [gds-types.h](#).

12.17.2 Field Documentation

12.17.2.1 x

```
int x
```

Definition at line [64](#) of file [gds-types.h](#).

12.17.2.2 y

```
int y
```

Definition at line [65](#) of file [gds-types.h](#).

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

- [gds-types.h](#)

12.18 gds_time_field Struct Reference

Date information for cells and libraries.

```
#include <gds-types.h>
```

Data Fields

- uint16_t [year](#)
- uint16_t [month](#)
- uint16_t [day](#)
- uint16_t [hour](#)
- uint16_t [minute](#)
- uint16_t [second](#)

12.18.1 Detailed Description

Date information for cells and libraries.

Definition at line [86](#) of file [gds-types.h](#).

12.18.2 Field Documentation

12.18.2.1 day

uint16_t day

Definition at line [89](#) of file [gds-types.h](#).

12.18.2.2 hour

uint16_t hour

Definition at line [90](#) of file [gds-types.h](#).

12.18.2.3 minute

uint16_t minute

Definition at line [91](#) of file [gds-types.h](#).

12.18.2.4 month

uint16_t month

Definition at line [88](#) of file [gds-types.h](#).

12.18.2.5 second

```
uint16_t second
```

Definition at line 92 of file [gds-types.h](#).

12.18.2.6 year

```
uint16_t year
```

Definition at line 87 of file [gds-types.h](#).

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

- [gds-types.h](#)

12.19 layer_element_dnd_data Struct Reference

This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

```
#include <layer-element.h>
```

Data Fields

- `GtkTargetEntry *` [entries](#)
Array of target entries for the DnD operation.
- `int` [entry_count](#)
Count of elements in `layer_element_dnd_data::entries` array.
- `void(* drag_begin)(GtkWidget *, GdkDragContext *, gpointer)`
Callback function for `drag_begin` event.
- `void(* drag_data_get)(GtkWidget *, GdkDragContext *, GtkSelectionData *, guint, guint, gpointer)`
Callback function for `data_get` event.
- `void(* drag_end)(GtkWidget *, GdkDragContext *, gpointer)`
Callback function for `drag_end` event.

12.19.1 Detailed Description

This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

Definition at line 63 of file [layer-element.h](#).

12.19.2 Field Documentation

12.19.2.1 drag_begin

```
void(* drag_begin) (GtkWidget *, GdkDragContext *, gpointer)
```

Callback function for drag_begin event.

Definition at line 69 of file [layer-element.h](#).

12.19.2.2 drag_data_get

```
void(* drag_data_get) (GtkWidget *, GdkDragContext *, GtkSelectionData *, guint, guint, gpointer)
```

Callback function for data_get event.

Definition at line 71 of file [layer-element.h](#).

12.19.2.3 drag_end

```
void(* drag_end) (GtkWidget *, GdkDragContext *, gpointer)
```

Callback function for drag_end event.

Definition at line 73 of file [layer-element.h](#).

12.19.2.4 entries

```
GtkTargetEntry* entries
```

Array of target entries for the DnD operation.

Definition at line 65 of file [layer-element.h](#).

12.19.2.5 entry_count

```
int entry_count
```

Count of elements in [layer_element_dnd_data::entries](#) array.

Definition at line 67 of file [layer-element.h](#).

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

- [layer-element.h](#)

12.20 layer_info Struct Reference

Layer information.

```
#include <layer-info.h>
```

Data Fields

- int [layer](#)
Layer number.
- char * [name](#)
Layer name.
- int [stacked_position](#)
Position of layer in output.
- GdkRGBA [color](#)
RGBA color used to render this layer.

12.20.1 Detailed Description

Layer information.

This structs contains information on how to render a layer

Definition at line 36 of file [layer-info.h](#).

12.20.2 Field Documentation

12.20.2.1 color

```
GdkRGBA color
```

RGBA color used to render this layer.

Definition at line 41 of file [layer-info.h](#).

12.20.2.2 layer

```
int layer
```

Layer number.

Definition at line 38 of file [layer-info.h](#).

12.20.2.3 name

```
char* name
```

Layer name.

Definition at line 39 of file [layer-info.h](#).

12.20.2.4 stacked_position

```
int stacked_position
```

Position of layer in output.

Warning

This parameter is not used by any renderer so far

Note

Lower is bottom, higher is top

Definition at line 40 of file [layer-info.h](#).

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

- [layer-info.h](#)

12.21 render_settings Struct Reference

This struct holds the renderer configuration.

```
#include <conv-settings-dialog.h>
```

Data Fields

- double [scale](#)
Scale image down by this factor.
- enum [output_renderer](#) [renderer](#)
- gboolean [tex_pdf_layers](#)
- gboolean [tex_standalone](#)

12.21.1 Detailed Description

This struct holds the renderer configuration.

Definition at line 56 of file [conv-settings-dialog.h](#).

12.21.2 Field Documentation

12.21.2.1 `renderer`

```
enum output_renderer renderer
```

The renderer to use

Definition at line 58 of file [conv-settings-dialog.h](#).

12.21.2.2 `scale`

```
double scale
```

Scale image down by this factor.

Note

Used to keep image in bound of maximum coordinate limit

Definition at line 57 of file [conv-settings-dialog.h](#).

12.21.2.3 `tex_pdf_layers`

```
gboolean tex_pdf_layers
```

Create OCG layers when rendering with TikZ

Definition at line 59 of file [conv-settings-dialog.h](#).

12.21.2.4 `tex_standalone`

```
gboolean tex_standalone
```

Create a standalone compile TeX file

Definition at line 60 of file [conv-settings-dialog.h](#).

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

- [conv-settings-dialog.h](#)

12.22 tree_stores Struct Reference

```
#include <tree-store.h>
```

Data Fields

- GtkTreeView * [base_tree_view](#)
- GtkTreeStore * [base_store](#)
- GtkTreeModelFilter * [filter](#)
- GtkEntry * [search_entry](#)

12.22.1 Detailed Description

Definition at line 46 of file [tree-store.h](#).

12.22.2 Field Documentation

12.22.2.1 base_store

```
GtkTreeStore* base_store
```

Definition at line 48 of file [tree-store.h](#).

12.22.2.2 base_tree_view

```
GtkTreeView* base_tree_view
```

Definition at line 47 of file [tree-store.h](#).

12.22.2.3 filter

```
GtkTreeModelFilter* filter
```

Definition at line 49 of file [tree-store.h](#).

12.22.2.4 search_entry

```
GtkEntry* search_entry
```

Definition at line 50 of file [tree-store.h](#).

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

- [tree-store.h](#)

12.23 vector_2d Struct Reference

```
#include <vector-operations.h>
```

Data Fields

- double [x](#)
- double [y](#)

12.23.1 Detailed Description

Definition at line 37 of file [vector-operations.h](#).

12.23.2 Field Documentation

12.23.2.1 x

```
double x
```

Definition at line 38 of file [vector-operations.h](#).

12.23.2.2 y

```
double y
```

Definition at line 39 of file [vector-operations.h](#).

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

- [vector-operations.h](#)

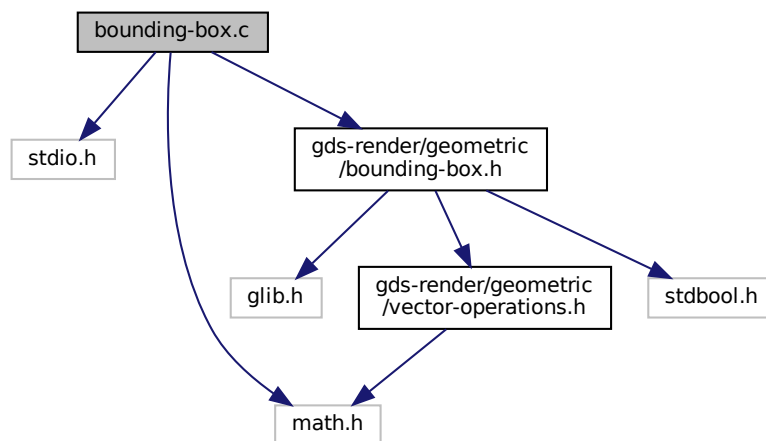
Chapter 13

File Documentation

13.1 bounding-box.c File Reference

Calculation of bounding boxes.

```
#include <stdio.h>
#include <math.h>
#include <gds-render/geometric/bounding-box.h>
Include dependency graph for bounding-box.c:
```



Macros

- `#define MIN(a, b) (((a) < (b)) ? (a) : (b))`
Return smaller number.
- `#define MAX(a, b) (((a) > (b)) ? (a) : (b))`
Return bigger number.
- `#define ABS_DBL(a) ((a) < 0 ? -(a) : (a))`

Functions

- void `bounding_box_calculate_polygon` (`GList *vertices`, `conv_generic_to_vector_2d_t conv_func`, `union bounding_box *box`)
- void `bounding_box_update_box` (`union bounding_box *destination`, `union bounding_box *update`)
- void `bounding_box_prepare_empty` (`union bounding_box *box`)
- static void `calculate_path_miter_points` (`struct vector_2d *a`, `struct vector_2d *b`, `struct vector_2d *c`, `struct vector_2d *m1`, `struct vector_2d *m2`, `double width`)
- void `bounding_box_calculate_path_box` (`GList *vertices`, `double thickness`, `conv_generic_to_vector_2d_t conv_func`, `union bounding_box *box`)
- void `bounding_box_update_point` (`union bounding_box *destination`, `conv_generic_to_vector_2d_t conv_func`, `void *pt`)
- void `bounding_box_apply_transform` (`double scale`, `double rotation_deg`, `bool flip_at_x`, `union bounding_box *box`)

Apply transformations onto bounding box.

13.1.1 Detailed Description

Calculation of bounding boxes.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file `bounding-box.c`.

13.2 bounding-box.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <stdio.h>
00032 #include <math.h>
00033
00034 #include <gds-render/geometric/bounding-box.h>
00035
00036 #define MIN(a,b) (((a) < (b)) ? (a) : (b))
00037 #define MAX(a,b) (((a) > (b)) ? (a) : (b))
00038 #define ABS_DBL(a) ((a) < 0 ? -(a) : (a))
00039
00040 void bounding_box_calculate_polygon(GList *vertices, conv_generic_to_vector_2d_t conv_func, union
    bounding_box *box)
00041 {
00042     double xmin = DBL_MAX, xmax = -DBL_MAX, ymin = DBL_MAX, ymax = -DBL_MAX;
00043     struct vector_2d temp_vec;
00044     GList *list_item;
00045
00046     /* Check for errors */
00047     if (!conv_func || !box || !vertices)
00048         return;

```

```

00049
00050     for (list_item = vertices; list_item != NULL; list_item = g_list_next(list_item)) {
00051         /* Convert generic vertex to vector_2d */
00052         if (conv_func)
00053             conv_func((void *)list_item->data, &temp_vec);
00054         else
00055             vector_2d_copy(&temp_vec, (struct vector_2d *)list_item->data);
00056
00057         /* Update bounding coordinates with vertex */
00058         xmin = MIN(xmin, temp_vec.x);
00059         xmax = MAX(xmax, temp_vec.x);
00060         ymin = MIN(ymin, temp_vec.y);
00061         ymax = MAX(ymax, temp_vec.y);
00062     }
00063
00064     /* Fill bounding box with results */
00065     box->vectors.lower_left.x = xmin;
00066     box->vectors.lower_left.y = ymin;
00067     box->vectors.upper_right.x = xmax;
00068     box->vectors.upper_right.y = ymax;
00069 }
00070
00071 void bounding_box_update_box(union bounding_box *destination, union bounding_box *update)
00072 {
00073     if (!destination || !update)
00074         return;
00075
00076     destination->vectors.lower_left.x = MIN(destination->vectors.lower_left.x,
00077         update->vectors.lower_left.x);
00078     destination->vectors.lower_left.y = MIN(destination->vectors.lower_left.y,
00079         update->vectors.lower_left.y);
00080     destination->vectors.upper_right.x = MAX(destination->vectors.upper_right.x,
00081         update->vectors.upper_right.x);
00082     destination->vectors.upper_right.y = MAX(destination->vectors.upper_right.y,
00083         update->vectors.upper_right.y);
00084 }
00085
00086 void bounding_box_prepare_empty(union bounding_box *box)
00087 {
00088     box->vectors.lower_left.x = DBL_MAX;
00089     box->vectors.lower_left.y = DBL_MAX;
00090     box->vectors.upper_right.x = -DBL_MAX;
00091     box->vectors.upper_right.y = -DBL_MAX;
00092 }
00093
00094 static void calculate_path_miter_points(struct vector_2d *a, struct vector_2d *b, struct vector_2d *c,
00095     struct vector_2d *m1, struct vector_2d *m2, double width)
00096 {
00097     double angle, angle_sin, u;
00098     struct vector_2d ba, bc, u_vec, v_vec, ba_norm;
00099
00100     if (!a || !b || !c || !m1 || !m2)
00101         return;
00102
00103     vector_2d_subtract(&ba, a, b);
00104     vector_2d_subtract(&bc, c, b);
00105
00106     angle = vector_2d_calculate_angle_between(&ba, &bc);
00107
00108     if (ABS_DBL(angle) < 0.05 || ABS_DBL(angle - M_PI) < 0.1) {
00109         /* Special cases Don*/
00110         vector_2d_copy(&ba_norm, &ba);
00111         vector_2d_rotate(&ba_norm, DEG2RAD(90));
00112         vector_2d_normalize(&ba_norm);
00113         vector_2d_scale(&ba_norm, width/2.0);
00114         vector_2d_add(m1, b, &ba_norm);
00115         vector_2d_subtract(m2, b, &ba_norm);
00116         return;
00117     }
00118     angle_sin = sin(angle);
00119     u = width/(2*angle_sin);
00120
00121     vector_2d_copy(&u_vec, &ba);
00122     vector_2d_copy(&v_vec, &bc);
00123     vector_2d_normalize(&u_vec);
00124     vector_2d_normalize(&v_vec);
00125     vector_2d_scale(&u_vec, u);
00126     vector_2d_scale(&v_vec, u);
00127
00128     vector_2d_copy(m1, b);
00129     vector_2d_add(m1, m1, &u_vec);
00130     vector_2d_add(m1, m1, &v_vec);
00131
00132     vector_2d_copy(m2, b);
00133     vector_2d_subtract(m2, m2, &u_vec);
00134     vector_2d_subtract(m2, m2, &v_vec);
00135 }

```

```

00136
00137 void bounding_box_calculate_path_box(GList *vertices, double thickness,
00138                                     conv_generic_to_vector_2d_t conv_func, union bounding_box *box)
00139 {
00140     GList *vertex_iterator;
00141     struct vector_2d pt;
00142
00143     printf("Warning! Function bounding_box_calculate_path_box not yet implemented correctly!\n");
00144
00145     if (!vertices || !box)
00146         return;
00147
00148     for (vertex_iterator = vertices; vertex_iterator != NULL; vertex_iterator =
00149         g_list_next(vertex_iterator)) {
00149
00150         if (conv_func != NULL)
00151             conv_func(vertex_iterator->data, &pt);
00152         else
00153             (void)vector_2d_copy(&pt, (struct vector_2d *)vertex_iterator->data);
00154
00155         /* These are approximations.
00156          * Used as long as miter point calculation is not fully implemented
00157          */
00158         box->vectors.lower_left.x = MIN(box->vectors.lower_left.x, pt.x - thickness/2);
00159         box->vectors.lower_left.y = MIN(box->vectors.lower_left.y, pt.y - thickness/2);
00160         box->vectors.upper_right.x = MAX(box->vectors.upper_right.x, pt.x + thickness/2);
00161         box->vectors.upper_right.y = MAX(box->vectors.upper_right.y, pt.y + thickness/2);
00162     }
00163 }
00164
00165 void bounding_box_update_point(union bounding_box *destination, conv_generic_to_vector_2d_t conv_func,
00166                               void *pt)
00167 {
00168     struct vector_2d point;
00169
00170     if (!destination || !pt)
00171         return;
00172
00173     if (conv_func)
00174         conv_func(pt, &point);
00175     else
00176         (void)vector_2d_copy(&point, (struct vector_2d *)pt);
00177
00178     destination->vectors.lower_left.x = MIN(destination->vectors.lower_left.x, point.x);
00179     destination->vectors.lower_left.y = MIN(destination->vectors.lower_left.y, point.y);
00180     destination->vectors.upper_right.x = MAX(destination->vectors.upper_right.x, point.x);
00181     destination->vectors.upper_right.y = MAX(destination->vectors.upper_right.y, point.y);
00182 }
00183
00190 void bounding_box_apply_transform(double scale, double rotation_deg, bool flip_at_x, union
00191                                 bounding_box *box)
00192 {
00193     int i;
00194
00195     /* Due to linearity, the order of the operations does not matter.
00196      * flip must be applied before rotation as defined by the GDS format
00197      */
00198     for (i = 0; i < 2; i++) {
00199         box->vector_array[i].y *= (flip_at_x ? -1 : 1);
00200         vector_2d_rotate(&box->vector_array[i], rotation_deg * M_PI / 180);
00201         vector_2d_scale(&box->vector_array[i], scale);
00202     }
00203 }

```

13.3 bounding-box.h File Reference

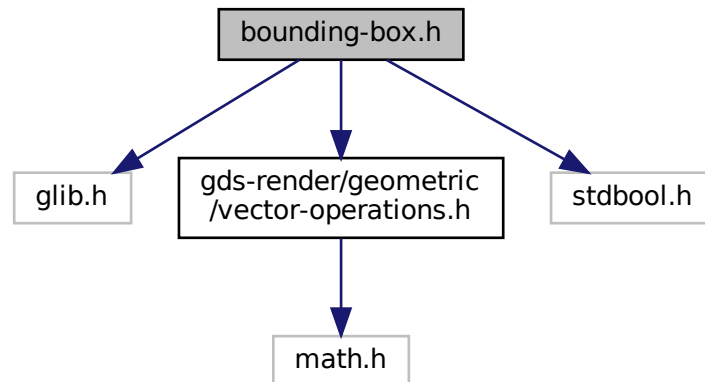
Header for calculation of bounding boxes.

```

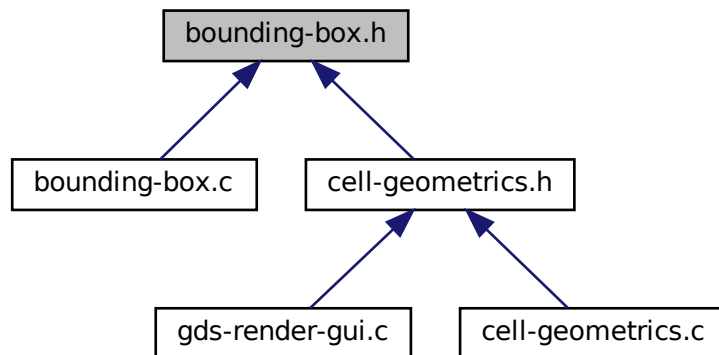
#include <glib.h>
#include <gds-render/geometric/vector-operations.h>
#include <stdbool.h>

```


Include dependency graph for bounding-box.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- union [bounding_box](#)
- struct [bounding_box::_vectors](#)

Typedefs

- typedef void(* [conv_generic_to_vector_2d_t](#)) (void *, struct [vector_2d](#) *)

Functions

- void `bounding_box_calculate_polygon` (GList *vertices, `conv_generic_to_vector_2d_t` conv_func, union `bounding_box` *box)
- void `bounding_box_update_box` (union `bounding_box` *destination, union `bounding_box` *update)
- void `bounding_box_prepare_empty` (union `bounding_box` *box)
- void `bounding_box_update_point` (union `bounding_box` *destination, `conv_generic_to_vector_2d_t` conv_↔ func, void *pt)
- void `bounding_box_apply_transform` (double scale, double rotation_deg, bool flip_at_x, union `bounding_box` *box)
 - Apply transformations onto bounding box.*
- void `bounding_box_calculate_path_box` (GList *vertices, double thickness, `conv_generic_to_vector_2d_t` conv_func, union `bounding_box` *box)

13.3.1 Detailed Description

Header for calculation of bounding boxes.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file `bounding-box.h`.

13.4 bounding-box.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _BOUNDING_BOX_H_
00032 #define _BOUNDING_BOX_H_
00033
00034 #include <glib.h>
00035 #include <gds-render/geometric/vector-operations.h>
00036 #include <stdbool.h>
00037
00038 union bounding_box {
00040     struct _vectors {
00041         struct vector_2d lower_left;
00042         struct vector_2d upper_right;
00043     } vectors;
00044     struct vector_2d vector_array[2];
00045 };
00046
00047 typedef void (*conv_generic_to_vector_2d_t)(void *, struct vector_2d *);
00048
00049 void bounding_box_calculate_polygon(GList *vertices, conv_generic_to_vector_2d_t conv_func, union
    bounding_box *box);
00050 void bounding_box_update_box(union bounding_box *destination, union bounding_box *update);
00051 void bounding_box_prepare_empty(union bounding_box *box);

```

```

00052 void bounding_box_update_point(union bounding\_box *destination, conv\_generic\_to\_vector\_2d\_t conv_func,
void *pt);
00053 void bounding_box_apply_transform(double scale, double rotation_deg, bool flip_at_x, union
bounding\_box *box);
00054 void bounding_box_calculate_path_box(GList *vertices, double thickness, conv\_generic\_to\_vector\_2d\_t
conv_func, union bounding\_box *box);
00055
00056 #endif /* _BOUNTING_BOX_H_ */
00057

```

13.5 cairo-output.c File Reference

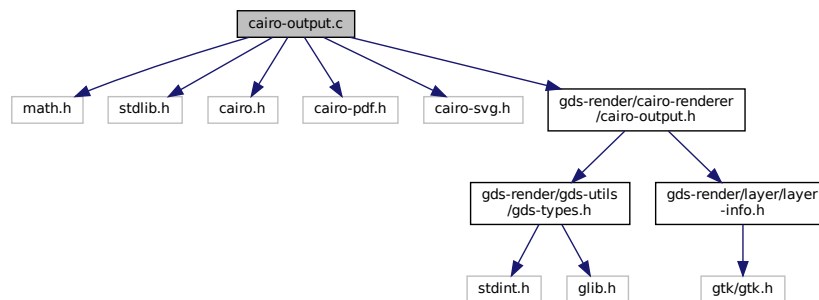
Output renderer for Cairo PDF export.

```

#include <math.h>
#include <stdlib.h>
#include <cairo.h>
#include <cairo-pdf.h>
#include <cairo-svg.h>
#include <gds-render/cairo-renderer/cairo-output.h>

```

Include dependency graph for cairo-output.c:



Data Structures

- struct [cairo_layer](#)

The [cairo_layer](#) struct Each rendered layer is represented by this struct.

Functions

- static void [revert_inherited_transform](#) (struct [cairo_layer](#) *layers)
 - Revert the last transformation on all layers.
- static void [apply_inherited_transform_to_all_layers](#) (struct [cairo_layer](#) *layers, const struct [gds_point](#) *origin, double magnification, gboolean flipping, double rotation, double scale)
 - Applies transformation to all layers.
- static void [render_cell](#) (struct [gds_cell](#) *cell, struct [cairo_layer](#) *layers, double scale)
 - render_cell Render a cell with its sub-cells
- void [cairo_render_cell_to_vector_file](#) (struct [gds_cell](#) *cell, [GList](#) *layer_infos, char *pdf_file, char *svg_file, double scale)
 - Render cell to a PDF file specified by pdf_file.

13.5.1 Detailed Description

Output renderer for Cairo PDF export.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [cairo-output.c](#).

13.6 cairo-output.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019 #include <math.h>
00020 #include <stdlib.h>
00021 #include <cairo.h>
00022 #include <cairo-pdf.h>
00023 #include <cairo-svg.h>
00024
00025 #include <gds-render/cairo-renderer/cairo-output.h>
00026
00027 struct cairo_layer {
00028     cairo_t *cr;
00029     cairo_surface_t *rec;
00030     struct layer_info *linfo;
00031 };
00032
00033 static void revert_inherited_transform(struct cairo_layer *layers)
00034 {
00035     int i;
00036
00037     for (i = 0; i < MAX_LAYERS; i++) {
00038         if (layers[i].cr == NULL)
00039             continue;
00040         cairo_restore(layers[i].cr);
00041     }
00042 }
00043
00044 static void apply_inherited_transform_to_all_layers(struct cairo_layer *layers,
00045                                                    const struct gds_point *origin,
00046                                                    double magnification,
00047                                                    gboolean flipping,
00048                                                    double rotation,
00049                                                    double scale)
00050 {
00051     int i;
00052     cairo_t *temp_layer_cr;
00053
00054     for (i = 0; i < MAX_LAYERS; i++) {
00055         temp_layer_cr = layers[i].cr;
00056         if (temp_layer_cr == NULL)
00057             continue;
00058
00059         /* Save the state and apply transformation */
00060         cairo_save(temp_layer_cr);
00061         cairo_translate(temp_layer_cr, (double)origin->x/scale, (double)origin->y/scale);
00062         cairo_rotate(temp_layer_cr, M_PI+rotation/180.0);
00063         cairo_scale(temp_layer_cr, magnification,
00064                   (flipping == TRUE ? -magnification : magnification));
00065     }
00066 }

```

```

00093 }
00094
00101 static void render_cell(struct gds_cell *cell, struct cairo_layer *layers, double scale)
00102 {
00103     GList *instance_list;
00104     struct gds_cell *temp_cell;
00105     struct gds_cell_instance *cell_instance;
00106     GList *gfx_list;
00107     struct gds_graphics *gfx;
00108     GList *vertex_list;
00109     struct gds_point *vertex;
00110     cairo_t *cr;
00111
00112     /* Render child cells */
00113     for (instance_list = cell->child_cells; instance_list != NULL; instance_list =
instance_list->next) {
00114         cell_instance = (struct gds_cell_instance *)instance_list->data;
00115         if ((temp_cell = cell_instance->cell_ref) != NULL) {
00116             apply_inherited_transform_to_all_layers(layers,
00117                 &cell_instance->origin,
00118                 cell_instance->magnification,
00119                 cell_instance->flipped,
00120                 cell_instance->angle,
00121                 scale);
00122             render_cell(temp_cell, layers, scale);
00123             revert_inherited_transform(layers);
00124         }
00125     }
00126
00127     /* Render graphics */
00128     for (gfx_list = cell->graphic_objs; gfx_list != NULL; gfx_list = gfx_list->next) {
00129         gfx = (struct gds_graphics *)gfx_list->data;
00130
00131         /* Get layer renderer */
00132         if (gfx->layer >= MAX_LAYERS)
00133             continue;
00134         if ((cr = layers[gfx->layer].cr) == NULL)
00135             continue;
00136
00137         /* Apply settings */
00138         cairo_set_line_width(cr, (gfx->width_absolute ? gfx->width_absolute/scale : 1));
00139
00140         switch (gfx->path_render_type) {
00141             case PATH_FLUSH:
00142                 cairo_set_line_cap(cr, CAIRO_LINE_CAP_BUTT);
00143                 break;
00144             case PATH_ROUNDED:
00145                 cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
00146                 break;
00147             case PATH_SQUARED:
00148                 cairo_set_line_cap(cr, CAIRO_LINE_CAP_SQUARE);
00149                 break;
00150         }
00151
00152         /* Add vertices */
00153         for (vertex_list = gfx->vertices; vertex_list != NULL; vertex_list = vertex_list->next) {
00154             vertex = (struct gds_point *)vertex_list->data;
00155
00156             /* If first point -> move to, else line to */
00157             if (vertex_list->prev == NULL)
00158                 cairo_move_to(cr, vertex->x/scale, vertex->y/scale);
00159             else
00160                 cairo_line_to(cr, vertex->x/scale, vertex->y/scale);
00161
00162         }
00163
00164         /* Create graphics object */
00165         switch (gfx->gfx_type) {
00166             case GRAPHIC_PATH:
00167                 cairo_stroke(cr);
00168                 break;
00169             case GRAPHIC_BOX:
00170             case GRAPHIC_POLYGON:
00171                 cairo_set_line_width(cr, 0.1/scale);
00172                 cairo_close_path(cr);
00173                 cairo_stroke_preserve(cr); // Prevent graphic glitches
00174                 cairo_fill(cr);
00175                 break;
00176         }
00177     }
00178 }
00179
00180 }
00181
00182 void cairo_render_cell_to_vector_file(struct gds_cell *cell, GList *layer_infos, char *pdf_file, char
*svg_file, double scale)
00183 {

```

```

00184     cairo_surface_t *pdf_surface, *svg_surface;
00185     cairo_t *pdf_cr, *svg_cr;
00186     struct layer_info *linfo;
00187     struct cairo_layer *layers;
00188     struct cairo_layer *lay;
00189     GList *info_list;
00190     int i;
00191     double rec_x0, rec_y0, rec_width, rec_height;
00192     double xmin = INT32_MAX, xmax = INT32_MIN, ymin = INT32_MAX, ymax = INT32_MIN;
00193
00194     if (pdf_file == NULL && svg_file == NULL) {
00195         /* No output specified */
00196         return;
00197     }
00198
00199     layers = (struct cairo_layer *)calloc(MAX_LAYERS, sizeof(struct cairo_layer));
00200
00201     /* Clear layers */
00202     for (i = 0; i < MAX_LAYERS; i++) {
00203         layers[i].cr = NULL;
00204         layers[i].rec = NULL;
00205     }
00206
00207     /* Create recording surface for each layer */
00208     for (info_list = layer_infos; info_list != NULL; info_list = g_list_next(info_list)) {
00209         linfo = (struct layer_info *)info_list->data;
00210         if (linfo->layer < MAX_LAYERS) {
00211             lay = &(layers[(unsigned int)linfo->layer]);
00212             lay->linfo = linfo;
00213             lay->rec = cairo_recording_surface_create(CAIRO_CONTENT_COLOR_ALPHA,
00214             NULL);
00215             lay->cr = cairo_create(layers[(unsigned int)linfo->layer].rec);
00216             cairo_scale(lay->cr, 1, -1); // Fix coordinate system
00217             cairo_set_source_rgb(lay->cr, linfo->color.red, linfo->color.green, linfo->color.blue);
00218         } else {
00219             printf("Layer number (%d) too high!\n", linfo->layer);
00220             goto ret_clear_layers;
00221         }
00222     }
00223
00224     render_cell(cell, layers, scale);
00225
00226     /* get size of image and top left coordinate */
00227     for (info_list = layer_infos; info_list != NULL; info_list = g_list_next(info_list)) {
00228         linfo = (struct layer_info *)info_list->data;
00229
00230         if (linfo->layer >= MAX_LAYERS) {
00231             printf("Layer outside of Spec.\n");
00232             continue;
00233         }
00234     }
00235
00236     /* Print size */
00237     cairo_recording_surface_ink_extents(layers[linfo->layer].rec, &rec_x0, &rec_y0,
00238     &rec_width, &rec_height);
00239     printf("Size of layer %d%s%s%s: <%lf x %lf> @ (%lf | %lf)\n",
00240     linfo->layer,
00241     (linfo->name && linfo->name[0] ? " (" : ""),
00242     (linfo->name && linfo->name[0] ? linfo->name : ""),
00243     (linfo->name && linfo->name[0] ? ")" : ""),
00244     rec_width, rec_height, rec_x0, rec_y0);
00245
00246     /* update bounding box */
00247     xmin = MIN(xmin, rec_x0);
00248     xmax = MAX(xmax, rec_x0);
00249     ymin = MIN(ymin, rec_y0);
00250     ymax = MAX(ymax, rec_y0);
00251     xmin = MIN(xmin, rec_x0+rec_width);
00252     xmax = MAX(xmax, rec_x0+rec_width);
00253     ymin = MIN(ymin, rec_y0+rec_height);
00254     ymax = MAX(ymax, rec_y0+rec_height);
00255
00256     }
00257
00258     printf("Cell bounding box: (%lf | %lf) -- (%lf | %lf)\n", xmin, ymin, xmax, ymax);
00259
00260     if (pdf_file) {
00261         pdf_surface = cairo_pdf_surface_create(pdf_file, xmax-xmin, ymax-ymin);
00262         pdf_cr = cairo_create(pdf_surface);
00263     }
00264
00265     if (svg_file) {
00266         svg_surface = cairo_svg_surface_create(svg_file, xmax-xmin, ymax-ymin);
00267         svg_cr = cairo_create(svg_surface);
00268     }
00269
00270     /* Write layers to PDF */

```

```

00271     for (info_list = layer_infos; info_list != NULL; info_list = g_list_next(info_list)) {
00272         linfo = (struct layer_info *)info_list->data;
00273
00274         if (linfo->layer >= MAX_LAYERS) {
00275             printf("Layer outside of Spec.\n");
00276             continue;
00277         }
00278
00279         if (pdf_file) {
00280             cairo_set_source_surface(pdf_cr, layers[linfo->layer].rec, -xmin, -ymin);
00281             cairo_paint_with_alpha(pdf_cr, linfo->color.alpha);
00282         }
00283
00284         if (svg_file) {
00285             cairo_set_source_surface(svg_cr, layers[linfo->layer].rec, -xmin, -ymin);
00286             cairo_paint_with_alpha(svg_cr, linfo->color.alpha);
00287         }
00288     }
00289 }
00290
00291 if (pdf_file) {
00292     cairo_show_page(pdf_cr);
00293     cairo_destroy(pdf_cr);
00294     cairo_surface_destroy(pdf_surface);
00295 }
00296
00297 if (svg_file) {
00298     cairo_show_page(svg_cr);
00299     cairo_destroy(svg_cr);
00300     cairo_surface_destroy(svg_surface);
00301 }
00302
00303 ret_clear_layers:
00304 for (i = 0; i < MAX_LAYERS; i++) {
00305     lay = &layers[i];
00306     if (lay->cr) {
00307         cairo_destroy(lay->cr);
00308         cairo_surface_destroy(lay->rec);
00309     }
00310 }
00311 free(layers);
00312
00313 printf("cairo export finished. It might still be buggy!\n");
00314 }
00315

```

13.7 cairo-output.h File Reference

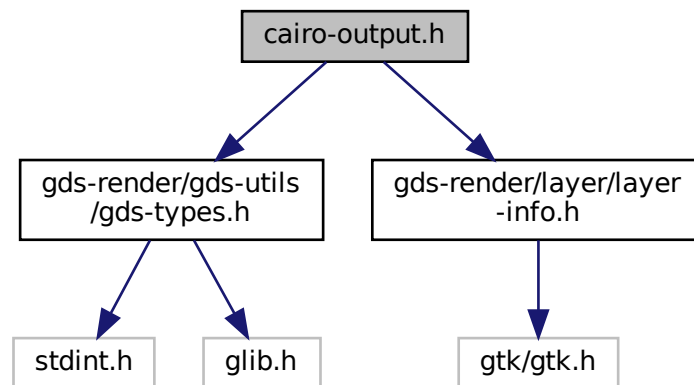
Header File for Cairo output renderer.

```

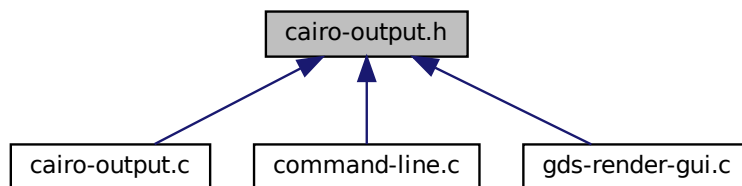
#include <gds-render/gds-utils/gds-types.h>
#include <gds-render/layer/layer-info.h>

```

Include dependency graph for `cairo-output.h`:



This graph shows which files directly or indirectly include this file:



Macros

- `#define MAX_LAYERS (300)`

Maximum layer count the output renderer can process. Typically GDS only specifies up to 255 layers.

Functions

- void `cairo_render_cell_to_vector_file` (struct `gds_cell` *cell, GList *layer_infos, char *pdf_file, char *svg_file, double scale)

Render cell to a PDF file specified by pdf_file.

13.7.1 Detailed Description

Header File for Cairo output renderer.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file `cairo-output.h`.

13.8 cairo-output.h

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00024 #ifndef _CAIRO_OUTPUT_H_
00025 #define _CAIRO_OUTPUT_H_
00026
00027 #include <gds-render/gds-utils/gds-types.h>
00028 #include <gds-render/layer/layer-info.h>
00029
00034 #define MAX_LAYERS (300)
00044 void cairo_render_cell_to_vector_file(struct gds_cell *cell, GList *layer_infos, char *pdf_file, char
    *svg_file, double scale);
00045
00048 #endif /* _CAIRO_OUTPUT_H_ */
```

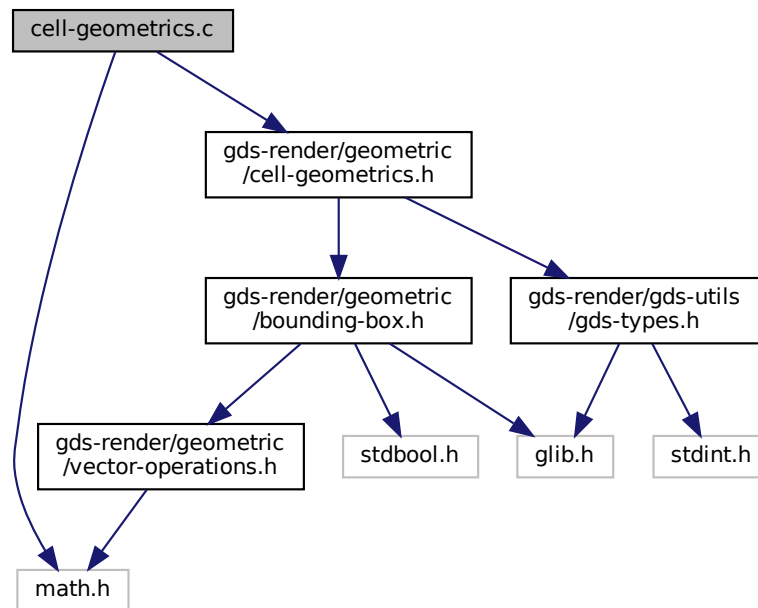
13.9 cairo-renderer.dox File Reference

13.10 cell-geometrics.c File Reference

Calculation of [gds_cell](#) trigonometrics.

```
#include <math.h>
#include <gds-render/geometric/cell-geometrics.h>
```

Include dependency graph for cell-geometrics.c:



Functions

- static void `convert_gds_point_to_2d_vector` (struct `gds_point` *pt, struct `vector_2d` *vector)
- static void `update_box_with_gfx` (union `bounding_box` *box, struct `gds_graphics` *gfx)
Update the given bounding box with the bounding box of a graphics element.
- void `calculate_cell_bounding_box` (union `bounding_box` *box, struct `gds_cell` *cell)
calculate_cell_bounding_box Calculate bounding box of gds cell

13.10.1 Detailed Description

Calculation of `gds_cell` trigonometrics.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [cell-geometrics.c](#).

13.11 cell-geometrics.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <math.h>
00027
00028 #include <gds-render/geometric/cell-geometrics.h>
00029
00035 static void convert_gds_point_to_2d_vector(struct gds_point *pt, struct vector_2d *vector)
00036 {
00037     vector->x = pt->x;
00038     vector->y = pt->y;
00039 }
00040
00046 static void update_box_with_gfx(union bounding_box *box, struct gds_graphics *gfx)
00047 {
00048     union bounding_box current_box;
00049
00050     bounding_box_prepare_empty(&current_box);
00051
00052     switch (gfx->gfx_type) {
00053     case GRAPHIC_BOX:
00054         /* Expected fallthrough */
00055     case GRAPHIC_POLYGON:
00056         bounding_box_calculate_polygon(gfx->vertices,
00057                                     (conv_generic_to_vector_2d_t)&convert_gds_point_to_2d_vector,
00058                                     &current_box);
00059         break;
00060     case GRAPHIC_PATH:
00061         /*
00062          * This is not implemented correctly.
00063          * Please be aware if paths are the outmost elements of your cell.
00064          * You might end up with a completely wrong calculated cell size.
00065          */
00066         bounding_box_calculate_path_box(gfx->vertices, gfx->width_absolute,
00067                                       (conv_generic_to_vector_2d_t)&convert_gds_point_to_2d_vector,
00068                                       &current_box);
00069         break;
00070     default:
00071         /* Unknown graphics object. */
00072         /* Print error? Nah.. */
00073         break;
00074     }
00075
00076     /* Update box with results */
00077     bounding_box_update_box(box, &current_box);
00078 }
00079
00080 void calculate_cell_bounding_box(union bounding_box *box, struct gds_cell *cell)
00081 {
00082     GList *gfx_list;
00083     struct gds_graphics *gfx;
00084     GList *sub_cell_list;
00085     struct gds_cell_instance *sub_cell;
00086     union bounding_box temp_box;
00087
00088     if (!box || !cell)
00089         return;
00090
00091     /* Update box with graphic elements */
00092     for (gfx_list = cell->graphic_objs; gfx_list != NULL; gfx_list = gfx_list->next) {
00093         gfx = (struct gds_graphics *)gfx_list->data;
00094         update_box_with_gfx(box, gfx);
00095     }
00096
00097     /* Update bounding box with boxes of subcells */
00098     for (sub_cell_list = cell->child_cells; sub_cell_list != NULL;
00099         sub_cell_list = sub_cell_list->next) {
00100         sub_cell = (struct gds_cell_instance *)sub_cell_list->data;
00101         bounding_box_prepare_empty(&temp_box);

```

```

00102     /* Recursion Woohoo!! This dies if your GDS is faulty and contains a reference loop */
00103     calculate_cell_bounding_box(&temp_box, sub_cell->cell_ref);
00104
00105     /* Apply transformations */
00106     bounding_box_apply_transform(ABS(sub_cell->magnification), sub_cell->angle,
00107                               sub_cell->flipped, &temp_box);
00108
00109     /* Move bounding box to origin */
00110     temp_box.vectors.lower_left.x += sub_cell->origin.x;
00111     temp_box.vectors.upper_right.x += sub_cell->origin.x;
00112     temp_box.vectors.lower_left.y += sub_cell->origin.y;
00113     temp_box.vectors.upper_right.y += sub_cell->origin.y;
00114
00115     /* update the parent's box */
00116     bounding_box_update_box(box, &temp_box);
00117 }
00118 }
00119

```

13.12 cell-geometrics.h File Reference

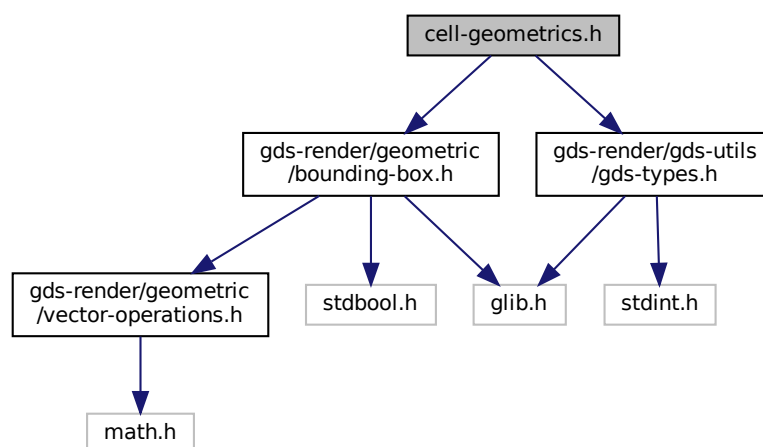
Calculation of `gds_cell` geometrics.

```

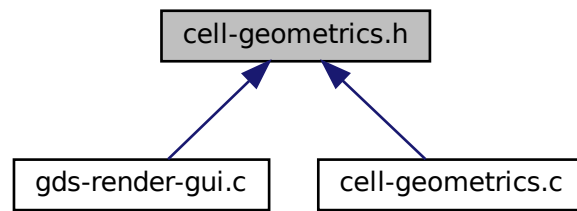
#include <gds-render/geometric/bounding-box.h>
#include <gds-render/gds-utils/gds-types.h>

```

Include dependency graph for `cell-geometrics.h`:



This graph shows which files directly or indirectly include this file:



Functions

- void `calculate_cell_bounding_box` (union `bounding_box` *box, struct `gds_cell` *cell)
calculate_cell_bounding_box Calculate bounding box of gds cell

13.12.1 Detailed Description

Calculation of `gds_cell` geometrics.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [cell-geometrics.h](#).

13.13 cell-geometrics.h

```

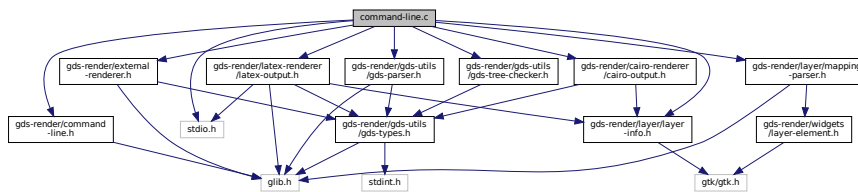
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _CELL_GEOMETRICS_H_
00032 #define _CELL_GEOMETRICS_H_
00033
00034 #include <gds-render/geometric/bounding-box.h>
00035 #include <gds-render/gds-utils/gds-types.h>
00036
00043 void calculate_cell_bounding_box(union bounding_box *box, struct gds_cell *cell);
00044
00045 #endif /* _CELL_GEOMETRICS_H_ */
00046
  
```

13.14 command-line.c File Reference

Function to render according to command line parameters.

```
#include <stdio.h>
#include <gds-render/command-line.h>
#include <gds-render/gds-utils/gds-parser.h>
#include <gds-render/layer/mapping-parser.h>
#include <gds-render/layer/layer-info.h>
#include <gds-render/cairo-renderer/cairo-output.h>
#include <gds-render/latex-renderer/latex-output.h>
#include <gds-render/external-renderer.h>
#include <gds-render/gds-utils/gds-tree-checker.h>
```

Include dependency graph for command-line.c:



Functions

- static void `delete_layer_info_with_name` (struct `layer_info` *info)
Delete `layer_info` and free `nem` element.
- void `command_line_convert_gds` (char *gds_name, char *pdf_name, char *tex_name, gboolean pdf, gboolean tex, char *layer_file, char *cell_name, double scale, gboolean pdf_layers, gboolean pdf_↔standalone, gboolean svg, char *svg_name, char *so_name, char *so_out_file)
Convert GDS according to supplied parameters.

13.14.1 Detailed Description

Function to render according to command line parameters.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [command-line.c](#).

13.15 command-line.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <stdio.h>
00032
00033 #include <gds-render/command-line.h>
00034 #include <gds-render/gds-utils/gds-parser.h>
00035 #include <gds-render/layer/mapping-parser.h>
00036 #include <gds-render/layer/layer-info.h>
00037 #include <gds-render/cairo-renderer/cairo-output.h>
00038 #include <gds-render/latex-renderer/latex-output.h>
00039 #include <gds-render/external-renderer.h>
00040 #include <gds-render/gds-utils/gds-tree-checker.h>
00041
00049 static void delete_layer_info_with_name(struct layer_info *info)
00050 {
00051     if (info) {
00052         if (info->name)
00053             g_free(info->name);
00054         free(info);
00055     }
00056 }
00057
00058 void command_line_convert_gds(char *gds_name, char *pdf_name, char *tex_name, gboolean pdf, gboolean
tex,
00059                             char *layer_file, char *cell_name, double scale, gboolean pdf_layers,
00060                             gboolean pdf_standalone, gboolean svg, char *svg_name, char *so_name, char
*so_out_file)
00061 {
00062     GList *libs = NULL;
00063     FILE *tex_file;
00064     int res;
00065     GFile *file;
00066     int i;
00067     GFileInputStream *stream;
00068     GDataInputStream *dstream;
00069     gboolean layer_export;
00070     GdkRGBA layer_color;
00071     int layer;
00072     char *layer_name;
00073     GList *layer_info_list = NULL;
00074     GList *cell_list;
00075     struct layer_info *linfo_temp;
00076     struct gds_library *first_lib;
00077     struct gds_cell *toplevel_cell = NULL, *temp_cell;
00078
00079
00080     /* Check if parameters are valid */
00081     if (!gds_name || (!pdf_name && pdf) || (!tex_name && tex) || !layer_file || !cell_name) {
00082         printf("Probably missing argument. Check --help option\n");
00083         return;
00084     }
00085
00086     /* Load GDS */
00087     clear_lib_list(&libs);
00088     res = parse_gds_from_file(gds_name, &libs);
00089     if (res)
00090         goto ret_destroy_library_list;
00091
00092     file = g_file_new_for_path(layer_file);
00093     stream = g_file_read(file, NULL, NULL);
00094
00095     if (!stream) {
00096         printf("Layer mapping not readable!\n");
00097         goto ret_destroy_file;
00098     }
00099     dstream = g_data_input_stream_new(G_INPUT_STREAM(stream));
00100     i = 0;
00101     do {

```

```

00102     res = mapping_parser_load_line(dstream, &layer_export, &layer_name, &layer, &layer_color);
00103     if (res == 0) {
00104         if (!layer_export)
00105             continue;
00106         linfo_temp = (struct layer_info *)malloc(sizeof(struct layer_info));
00107         if (!linfo_temp) {
00108             printf("Out of memory\n");
00109             goto ret_clear_layer_list;
00110         }
00111         linfo_temp->color.alpha = layer_color.alpha;
00112         linfo_temp->color.red = layer_color.red;
00113         linfo_temp->color.green = layer_color.green;
00114         linfo_temp->color.blue = layer_color.blue;
00115         linfo_temp->name = layer_name;
00116         linfo_temp->stacked_position = i++;
00117         linfo_temp->layer = layer;
00118         layer_info_list = g_list_append(layer_info_list, (gpointer)linfo_temp);
00119     }
00120 } while(res >= 0);
00121
00122
00123 /* find_cell in first library. */
00124 if (!libs)
00125     goto ret_clear_layer_list;
00126
00127 first_lib = (struct gds_library *)libs->data;
00128 if (!first_lib) {
00129     fprintf(stderr, "No library in library list. This should not happen.\n");
00130     goto ret_clear_layer_list;
00131 }
00132
00133 for (cell_list = first_lib->cells; cell_list != NULL; cell_list = g_list_next(cell_list)) {
00134     temp_cell = (struct gds_cell *)cell_list->data;
00135     if (!strcmp(temp_cell->name, cell_name)) {
00136         toplevel_cell = temp_cell;
00137         break;
00138     }
00139 }
00140
00141 if (!toplevel_cell) {
00142     printf("Couldn't find cell in first library!\n");
00143     goto ret_clear_layer_list;
00144 }
00145
00146 /* Check if cell passes vital checks */
00147 res = gds_tree_check_reference_loops(toplevel_cell->parent_library);
00148 if (res < 0) {
00149     fprintf(stderr, "Checking library %s failed.\n", first_lib->name);
00150     goto ret_clear_layer_list;
00151 } else if (res > 0) {
00152     fprintf(stderr, "%d reference loops found.\n", res);
00153
00154     /* do further checking if the specified cell and/or its subcells are affected */
00155     if (toplevel_cell->checks.affected_by_reference_loop == 1) {
00156         fprintf(stderr, "Cell is affected by reference loop. Abort!\n");
00157         goto ret_clear_layer_list;
00158     }
00159 }
00160
00161 if (toplevel_cell->checks.affected_by_reference_loop == GDS_CELL_CHECK_NOT_RUN)
00162     fprintf(stderr, "Cell was not checked. This should not happen. Please report this issue. Will
continue either way.\n");
00163
00164 /* Note: unresolved references are not an abort condition.
00165 * Deal with it.
00166 */
00167
00168 /* Render outputs */
00169 if (pdf == TRUE || svg == TRUE) {
00170     cairo_render_cell_to_vector_file(toplevel_cell, layer_info_list, (pdf == TRUE ? pdf_name :
NULL),
00171                                     (svg == TRUE ? svg_name : NULL), scale);
00172 }
00173
00174 if (tex == TRUE) {
00175     tex_file = fopen(tex_name, "w");
00176     if (!tex_file)
00177         goto ret_clear_layer_list;
00178     latex_render_cell_to_code(toplevel_cell, layer_info_list, tex_file, scale, pdf_layers,
pdf_standalone);
00179     fclose(tex_file);
00180 }
00181
00182 if (so_name && so_out_file) {
00183     if (strlen(so_name) == 0 || strlen(so_out_file) == 0)
00184         goto ret_clear_layer_list;
00185 }

```



```
00186     /* Render output using external renderer */
00187     printf("Invoking external renderer!\n");
00188     external_renderer_render_cell(toplevel_cell, layer_info_list, so_out_file, so_name);
00189     printf("External renderer finished!\n");
00190 }
00191
00192 ret_clear_layer_list:
00193     g_list_free_full(layer_info_list, (GDestroyNotify)delete_layer_info_with_name);
00194
00195     g_object_unref(dstream);
00196     g_object_unref(stream);
00197 ret_destroy_file:
00198     g_object_unref(file);
00199     /* Delete all allocated libraries */
00200 ret_destroy_library_list:
00201     clear_lib_list(&libs);
00202 }
00203
```

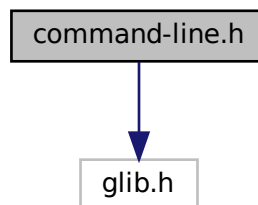
13.16 command-line.dox File Reference

13.17 command-line.h File Reference

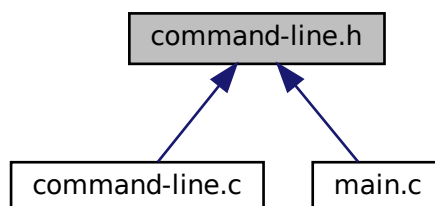
Render according to command line parameters.

```
#include <glib.h>
```

Include dependency graph for command-line.h:



This graph shows which files directly or indirectly include this file:



Functions

- void `command_line_convert_gds` (char *gds_name, char *pdf_name, char *tex_name, gboolean pdf, gboolean tex, char *layer_file, char *cell_name, double scale, gboolean pdf_layers, gboolean pdf_↔standalone, gboolean svg, char *svg_name, char *so_name, char *so_out_file)

Convert GDS according to supplied parameters.

13.17.1 Detailed Description

Render according to command line parameters.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [command-line.h](#).

13.18 command-line.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _COMMAND_LINE_H_
00032 #define _COMMAND_LINE_H_
00033
00034 #include <glib.h>
00035
00055 void command_line_convert_gds(char *gds_name, char *pdf_name, char *tex_name, gboolean pdf, gboolean
tex,
00056                               char *layer_file, char *cell_name, double scale, gboolean pdf_layers,
00057                               gboolean pdf_standalone, gboolean svg, char *svg_name, char *so_name, char
*so_out_file);
00058
00059 #endif /* _COMMAND_LINE_H_ */
00060

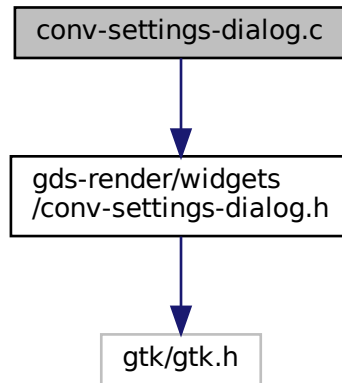
```

13.19 compilation.dox File Reference

13.20 conv-settings-dialog.c File Reference

Implementation of the setting dialog.

```
#include <gds-render/widgets/conv-settings-dialog.h>
Include dependency graph for conv-settings-dialog.c:
```



Data Structures

- struct [_RendererSettingsDialog](#)

Enumerations

- enum { [PROP_CELL_NAME](#) = 1, [PROP_COUNT](#) }

Functions

- static void [renderer_settings_dialog_set_property](#) (GObject *object, guint property_id, const GValue *value, GParamSpec *pspec)
- static void [renderer_settings_dialog_get_property](#) (GObject *object, guint property_id, GValue *value, GParamSpec *pspec)
- static void [renderer_settings_dialog_class_init](#) (RendererSettingsDialogClass *klass)
- static void [show_tex_options](#) (RendererSettingsDialog *self)
- static void [hide_tex_options](#) (RendererSettingsDialog *self)
- static void [latex_render_callback](#) (GtkToggleButton *radio, RendererSettingsDialog *dialog)
- static gboolean [shape_drawer_drawing_callback](#) (GtkWidget *widget, cairo_t *cr, gpointer data)
- static double [convert_number_to_engineering](#) (double input, const char **out_prefix)
- static void [renderer_settings_dialog_update_labels](#) (RendererSettingsDialog *self)
- static void [scale_value_changed](#) (GtkRange *range, gpointer user_data)
- static void [renderer_settings_dialog_init](#) (RendererSettingsDialog *self)
- RendererSettingsDialog * [renderer_settings_dialog_new](#) (GtkWindow *parent)
 - *Create a new RedererSettingsDialog GObject.*
- void [renderer_settings_dialog_get_settings](#) (RendererSettingsDialog *dialog, struct [render_settings](#) *settings)
 - *Get the settings configured in the dialog.*

- G_END_DECLS void `renderer_settings_dialog_set_settings` (RendererSettingsDialog *dialog, struct `render_settings` *settings)
Apply settings to dialog.
- void `renderer_settings_dialog_set_cell_width` (RendererSettingsDialog *dialog, unsigned int width)
renderer_settings_dialog_set_cell_width Set width for rendered cell
- void `renderer_settings_dialog_set_cell_height` (RendererSettingsDialog *dialog, unsigned int height)
renderer_settings_dialog_set_cell_height Set height for rendered cell
- void `renderer_settings_dialog_set_database_unit_scale` (RendererSettingsDialog *dialog, double unit_in_meters)
renderer_settings_dialog_set_database_unit_scale Set database scale

Variables

- static GParamSpec * `properties` [PROP_COUNT]

13.20.1 Detailed Description

Implementation of the setting dialog.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file `conv-settings-dialog.c`.

13.21 conv-settings-dialog.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #include <gds-render/widgets/conv-settings-dialog.h>
00033
00034 struct _RendererSettingsDialog {
00035     GtkDialog parent;
00036     /* Private loot */
00037     GtkWidget *radio_latex;
00038     GtkWidget *radio_cairo_pdf;
00039     GtkWidget *radio_cairo_svg;
00040     GtkWidget *scale;
00041     GtkWidget *layer_check;
00042     GtkWidget *standalone_check;
00043     GtkDrawingArea *shape_drawing;
00044     GtkLabel *x_label;
00045     GtkLabel *y_label;
00046
00047     GtkLabel *x_output_label;
00048     GtkLabel *y_output_label;
00049

```

```

00050     unsigned int cell_height;
00051     unsigned int cell_width;
00052     double unit_in_meters;
00053 };
00054
00055 G_DEFINE_TYPE(RendererSettingsDialog, renderer_settings_dialog, GTK_TYPE_DIALOG)
00056
00057 enum {
00058     PROP_CELL_NAME = 1,
00059     PROP_COUNT
00060 };
00061
00062 static GParamSpec *properties[PROP_COUNT];
00063
00064 static void renderer_settings_dialog_set_property(GObject *object, guint property_id,
00065     const GValue *value, GParamSpec *pspec)
00066 {
00067     const gchar *title = NULL;
00068
00069     switch (property_id) {
00070     case PROP_CELL_NAME:
00071         title = g_value_get_string(value);
00072         if (title)
00073             gtk_window_set_title(GTK_WINDOW(object), title);
00074         break;
00075     default:
00076         G_OBJECT_WARN_INVALID_PROPERTY_ID(object, property_id, pspec);
00077         break;
00078     }
00079 }
00080
00081 static void renderer_settings_dialog_get_property(GObject *object, guint property_id,
00082     GValue *value, GParamSpec *pspec)
00083 {
00084     const gchar *title;
00085
00086     switch (property_id) {
00087     case PROP_CELL_NAME:
00088         title = gtk_window_get_title(GTK_WINDOW(object));
00089         g_value_set_string(value, title);
00090         break;
00091     default:
00092         G_OBJECT_WARN_INVALID_PROPERTY_ID(object, property_id, pspec);
00093         break;
00094     }
00095 }
00096
00097 static void renderer_settings_dialog_class_init(RendererSettingsDialogClass *klass)
00098 {
00099     GObjectClass *oclass = G_OBJECT_CLASS(klass);
00100
00101     /* Override virtual functions */
00102     oclass->set_property = renderer_settings_dialog_set_property;
00103     oclass->get_property = renderer_settings_dialog_get_property;
00104
00105     properties[PROP_CELL_NAME] = g_param_spec_string("cell-name",
00106         "cell-name",
00107         "Cell name to be displayed in header bar",
00108         "",
00109         G_PARAM_READWRITE);
00110     g_object_class_install_properties(oclass, PROP_COUNT, properties);
00111 }
00112
00113 static void show_tex_options(RendererSettingsDialog *self)
00114 {
00115     gtk_widget_show(self->layer_check);
00116     gtk_widget_show(self->standalone_check);
00117 }
00118 }
00119
00120 static void hide_tex_options(RendererSettingsDialog *self)
00121 {
00122     gtk_widget_hide(self->layer_check);
00123     gtk_widget_hide(self->standalone_check);
00124 }
00125
00126 static void latex_render_callback(GtkToggleButton *radio, RendererSettingsDialog *dialog)
00127 {
00128     if (gtk_toggle_button_get_active(radio))
00129         show_tex_options(dialog);
00130     else
00131         hide_tex_options(dialog);
00132 }
00133
00134 static gboolean shape_drawer_drawing_callback(GtkWidget *widget, cairo_t *cr, gpointer data)
00135 {
00136     int width;

```

```

00137     int height;
00138     GtkStyleContext *style_context;
00139     GdkRGBA foreground_color;
00140     RendererSettingsDialog *dialog = (RendererSettingsDialog *)data;
00141     double usable_width;
00142     double usable_height;
00143     double height_scale;
00144     double width_scale;
00145     double final_scale_value;
00146
00147     style_context = gtk_widget_get_style_context(widget);
00148     width = gtk_widget_get_allocated_width(widget);
00149     height = gtk_widget_get_allocated_height(widget);
00150
00151     gtk_render_background(style_context, cr, 0, 0, width, height);
00152
00153     gtk_style_context_get_color(style_context, gtk_style_context_get_state(style_context),
00154                               &foreground_color);
00155
00156     gdk_cairo_set_source_rgba(cr, &foreground_color);
00157
00158     cairo_save(cr);
00159
00160     /* Transform coordiante system */
00161     cairo_scale(cr, 1, -1);
00162     cairo_translate(cr, (double)width/2.0, -(double)height/2.0);
00163
00164     /* Define usable drawing area */
00165     usable_width = (0.95*(double)width) - 15.0;
00166     usable_height = (0.95*(double)height) - 15.0;
00167
00168     width_scale = usable_width/(double)dialog->cell_width;
00169     height_scale = usable_height/(double)dialog->cell_height;
00170
00171     final_scale_value = (width_scale < height_scale ? width_scale : height_scale);
00172
00173     cairo_rectangle(cr, -(double)dialog->cell_width*final_scale_value/2.0,
00174                  -(double)dialog->cell_height*final_scale_value/2.0,
00175                  (double)dialog->cell_width*final_scale_value,
00176                  (double)dialog->cell_height*final_scale_value);
00177     cairo_stroke(cr);
00178     cairo_restore(cr);
00179 }
00180
00181 static double convert_number_to_engineering(double input, const char **out_prefix)
00182 {
00183     const char *selected_prefix = NULL;
00184     double return_val = 0.0;
00185     int idx;
00186     const static char * prefixes[] = {"y", "z", "a", "f", "p", "n", "u", "m", "c", "d", /* < 1 */
00187                                       "h", /* 1 */
00188                                       "k", "M", "G", "T", "P", "E", "Z", "Y"}; /* > 1 */
00189     const static double scale[] = {1E-24, 1E-21, 1E-18, 1E-15, 1E-12, 1E-9, 1E-6, 1E-3, 1E-2, 1E-1,
00190                                   1,
00191                                   1E2, 1E3, 1E6, 1E9, 1E12, 1E15, 1E18, 1E21, 1E24};
00192     const int prefix_count = (int)(sizeof(prefixes)/sizeof(char *));
00193
00194     /* If pointer is invalid, return NaN */
00195     if (!out_prefix)
00196         return 0.0 / 0.0;
00197
00198     /* Start with the 2nd smallest prefix */
00199     for (idx = 1; idx < prefix_count; idx++) {
00200         if (input < scale[idx]) {
00201             /* This prefix is bigger than the number. Take the previous one */
00202             selected_prefix = prefixes[idx-1];
00203             return_val = input / scale[idx-1];
00204             break;
00205         }
00206     }
00207
00208     /* Check if prefix was set by loop. Else take the largest in the list */
00209     if (selected_prefix == NULL) {
00210         selected_prefix = prefixes[prefix_count-1];
00211         return_val = input / scale[prefix_count-1];
00212     }
00213
00214     if (out_prefix)
00215         *out_prefix = selected_prefix;
00216
00217     return return_val;
00218 }
00219
00220 static void renderer_settings_dialog_update_labels(RendererSettingsDialog *self)
00221 {

```

```

00222     char default_buff[100];
00223     double scale;
00224     double width_meters;
00225     double height_meters;
00226     double width_engineering;
00227     const char *width_prefix;
00228     double height_engineering;
00229     const char *height_prefix;
00230
00231     if (!self)
00232         return;
00233
00234     width_meters = (double)self->cell_width * self->unit_in_meters;
00235     height_meters = (double)self->cell_height * self->unit_in_meters;
00236
00237     width_engineering = convert_number_to_engineering(width_meters, &width_prefix);
00238     height_engineering = convert_number_to_engineering(height_meters, &height_prefix);
00239
00240     snprintf(default_buff, sizeof(default_buff), "Width: %.3lf %sm", width_engineering, width_prefix);
00241     gtk_label_set_text(self->x_label, default_buff);
00242     snprintf(default_buff, sizeof(default_buff), "Height: %.3lf %sm", height_engineering,
height_prefix);
00243     gtk_label_set_text(self->y_label, default_buff);
00244
00245     scale = gtk_range_get_value(GTK_RANGE(self->scale));
00246
00247     /* Set the pixel sizes */
00248     snprintf(default_buff, sizeof(default_buff), "Output Width: %u px",
(unsigned int)((double)self->cell_width / scale));
00249     gtk_label_set_text(self->x_output_label, default_buff);
00250     snprintf(default_buff, sizeof(default_buff), "Output Height: %u px",
(unsigned int)((double)self->cell_height / scale));
00251     gtk_label_set_text(self->y_output_label, default_buff);
00252 }
00253
00254 static void scale_value_changed(GtkRange *range, gpointer user_data)
00255 {
00256     (void)range;
00257     RendererSettingsDialog *dialog;
00258
00259     dialog = RENDERER_SETTINGS_DIALOG(user_data);
00260     renderer_settings_dialog_update_labels(dialog);
00261 }
00262
00263 static void renderer_settings_dialog_init(RendererSettingsDialog *self)
00264 {
00265     GtkBuilder *builder;
00266     GtkWidget *box;
00267     GtkDialog *dialog;
00268
00269     dialog = &self->parent;
00270
00271     builder = gtk_builder_new_from_resource("/dialog.glade");
00272     box = GTK_WIDGET(gtk_builder_get_object(builder, "dialog-box"));
00273     self->radio_latex = GTK_WIDGET(gtk_builder_get_object(builder, "latex-radio"));
00274     self->radio_cairo_pdf = GTK_WIDGET(gtk_builder_get_object(builder, "cairo-pdf-radio"));
00275     self->radio_cairo_svg = GTK_WIDGET(gtk_builder_get_object(builder, "cairo-svg-radio"));
00276     self->scale = GTK_WIDGET(gtk_builder_get_object(builder, "dialog-scale"));
00277     self->standalone_check = GTK_WIDGET(gtk_builder_get_object(builder, "standalone-check"));
00278     self->layer_check = GTK_WIDGET(gtk_builder_get_object(builder, "layer-check"));
00279     self->shape_drawing = GTK_DRAWING_AREA(gtk_builder_get_object(builder, "shape-drawer"));
00280     self->x_label = GTK_LABEL(gtk_builder_get_object(builder, "x-label"));
00281     self->y_label = GTK_LABEL(gtk_builder_get_object(builder, "y-label"));
00282     self->x_output_label = GTK_LABEL(gtk_builder_get_object(builder, "x-output-label"));
00283     self->y_output_label = GTK_LABEL(gtk_builder_get_object(builder, "y-output-label"));
00284
00285     gtk_dialog_add_buttons(dialog, "Cancel", GTK_RESPONSE_CANCEL, "OK", GTK_RESPONSE_OK, NULL);
00286     gtk_container_add(GTK_CONTAINER(gtk_dialog_get_content_area(dialog)), box);
00287     gtk_window_set_title(GTK_WINDOW(self), "Renderer Settings");
00288
00289     g_signal_connect(self->radio_latex, "toggled", G_CALLBACK(latex_render_callback), (gpointer)self);
00290     g_signal_connect(G_OBJECT(self->shape_drawing),
"draw", G_CALLBACK(shape_drawer_drawing_callback), (gpointer)self);
00291
00292     g_signal_connect(self->scale, "value-changed", G_CALLBACK(scale_value_changed), (gpointer)self);
00293
00294     /* Default values */
00295     self->cell_width = 1;
00296     self->cell_height = 1;
00297     self->unit_in_meters = 1E-6;
00298     renderer_settings_dialog_update_labels(self);
00299
00300     g_object_unref(builder);
00301 }
00302
00303 RendererSettingsDialog *renderer_settings_dialog_new(GtkWindow *parent)
00304 {
00305 }

```

```

00308     RendererSettingsDialog *res;
00309
00310     res = RENDERER_SETTINGS_DIALOG(g_object_new(RENDERER_TYPE_SETTINGS_DIALOG, NULL));
00311     if (res && parent) {
00312         gtk_window_set_transient_for(GTK_WINDOW(res), parent);
00313     }
00314     return res;
00315 }
00316
00317 void renderer_settings_dialog_get_settings(RendererSettingsDialog *dialog, struct render_settings
    *settings)
00318 {
00319
00320     if (!settings || !dialog)
00321         return;
00322     settings->scale = gtk_range_get_value(GTK_RANGE(dialog->scale));
00323
00324     /* Get active radio button selection */
00325     if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->radio_latex)) == TRUE) {
00326         settings->renderer = RENDERER_LATEX_TIKZ;
00327     } else if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_pdf)) == TRUE) {
00328         settings->renderer = RENDERER_CAIROGRAPHICS_PDF;
00329     } else if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_svg)) == TRUE) {
00330         settings->renderer = RENDERER_CAIROGRAPHICS_SVG;
00331     }
00332
00333     settings->tex_pdf_layers = gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->layer_check));
00334     settings->tex_standalone =
    gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(dialog->standalone_check));
00335 }
00336
00337 void renderer_settings_dialog_set_settings(RendererSettingsDialog *dialog, struct render_settings
    *settings)
00338 {
00339     if (!settings || !dialog)
00340         return;
00341
00342     gtk_range_set_value(GTK_RANGE(dialog->scale), settings->scale);
00343     gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->layer_check), settings->tex_pdf_layers);
00344     gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->standalone_check),
    settings->tex_standalone);
00345
00346     switch (settings->renderer) {
00347     case RENDERER_LATEX_TIKZ:
00348         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->radio_latex), TRUE);
00349         show_tex_options(dialog);
00350         break;
00351     case RENDERER_CAIROGRAPHICS_PDF:
00352         hide_tex_options(dialog);
00353         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_pdf), TRUE);
00354         break;
00355     case RENDERER_CAIROGRAPHICS_SVG:
00356         hide_tex_options(dialog);
00357         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(dialog->radio_cairo_svg), TRUE);
00358         break;
00359     }
00360 }
00361
00362 void renderer_settings_dialog_set_cell_width(RendererSettingsDialog *dialog, unsigned int width)
00363 {
00364     if (!dialog)
00365         return;
00366
00367     if (width == 0)
00368         width = 1;
00369
00370     dialog->cell_width = width;
00371     renderer_settings_dialog_update_labels(dialog);
00372 }
00373
00374 void renderer_settings_dialog_set_cell_height(RendererSettingsDialog *dialog, unsigned int height)
00375 {
00376     if (!dialog)
00377         return;
00378
00379     if (height == 0)
00380         height = 1;
00381
00382     dialog->cell_height = height;
00383     renderer_settings_dialog_update_labels(dialog);
00384 }
00385
00386 void renderer_settings_dialog_set_database_unit_scale(RendererSettingsDialog *dialog, double
    unit_in_meters)
00387 {
00388     if (!dialog)
00389         return;

```



```

00390
00391     if (unit_in_meters < 0)
00392         unit_in_meters *= -1;
00393
00394     dialog->unit_in_meters = unit_in_meters;
00395     renderer_settings_dialog_update_labels(dialog);
00396 }
00397

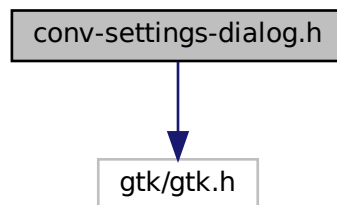
```

13.22 conv-settings-dialog.h File Reference

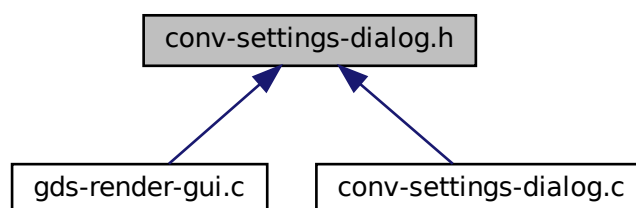
Header file for the Conversion Settings Dialog.

```
#include <gtk/gtk.h>
```

Include dependency graph for conv-settings-dialog.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [renderer_settings](#)

This struct holds the renderer configuration.

Macros

- #define [RENDERER_TYPE_SETTINGS_DIALOG](#) (renderer_settings_dialog_get_type())

Enumerations

- enum `output_renderer` { `RENDERER_LATEX_TIKZ`, `RENDERER_CAIROGRAPHICS_PDF`, `RENDERER_CAIROGRAPHICS_` }

return type of the `RendererSettingsDialog`

Functions

- `RendererSettingsDialog * renderer_settings_dialog_new` (`GtkWindow *parent`)
Create a new `RendererSettingsDialog` GObject.
- `G_END_DECLS void renderer_settings_dialog_set_settings` (`RendererSettingsDialog *dialog`, `struct render_settings *settings`)
Apply settings to dialog.
- `void renderer_settings_dialog_get_settings` (`RendererSettingsDialog *dialog`, `struct render_settings *settings`)
Get the settings configured in the dialog.
- `void renderer_settings_dialog_set_cell_width` (`RendererSettingsDialog *dialog`, `unsigned int width`)
renderer_settings_dialog_set_cell_width Set width for rendered cell
- `void renderer_settings_dialog_set_cell_height` (`RendererSettingsDialog *dialog`, `unsigned int height`)
renderer_settings_dialog_set_cell_height Set height for rendered cell
- `void renderer_settings_dialog_set_database_unit_scale` (`RendererSettingsDialog *dialog`, `double unit_in_↔ meters`)
renderer_settings_dialog_set_database_unit_scale Set database scale

13.22.1 Detailed Description

Header file for the Conversion Settings Dialog.

Author

`Mario.Huettel@gmx.net` `mario.huettel@gmx.net`

Definition in file `conv-settings-dialog.h`.

13.23 conv-settings-dialog.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef __CONV_SETTINGS_DIALOG_H__
00033 #define __CONV_SETTINGS_DIALOG_H__
00034

```

```

00035 #include <gtk/gtk.h>
00036
00037 G_BEGIN_DECLS
00038
00040 enum output_renderer {RENDERER_LATEX_TIKZ, RENDERER_CAIROGRAPHICS_PDF, RENDERER_CAIROGRAPHICS_SVG};
00041
00042 G_DECLARE_FINAL_TYPE(RendererSettingsDialog, renderer_settings_dialog, RENDERER, SETTINGS_DIALOG,
    GtkDialog)
00043
00044
00049 RendererSettingsDialog *renderer_settings_dialog_new(GtkWindow *parent);
00050
00051 #define RENDERER_TYPE_SETTINGS_DIALOG (renderer_settings_dialog_get_type())
00052
00056 struct render_settings {
00057     double scale;
00058     enum output_renderer renderer;
00059     gboolean tex_pdf_layers;
00060     gboolean tex_standalone;
00061 };
00062
00063 G_END_DECLS
00064
00070 void renderer_settings_dialog_set_settings(RendererSettingsDialog *dialog, struct render_settings
    *settings);
00071
00077 void renderer_settings_dialog_get_settings(RendererSettingsDialog *dialog, struct render_settings
    *settings);
00078
00084 void renderer_settings_dialog_set_cell_width(RendererSettingsDialog *dialog, unsigned int width);
00085
00091 void renderer_settings_dialog_set_cell_height(RendererSettingsDialog *dialog, unsigned int height);
00092
00098 void renderer_settings_dialog_set_database_unit_scale(RendererSettingsDialog *dialog, double
    unit_in_meters);
00099
00100 #endif /* __CONV_SETTINGS_DIALOG_H__ */
00101

```

13.24 external-renderer.c File Reference

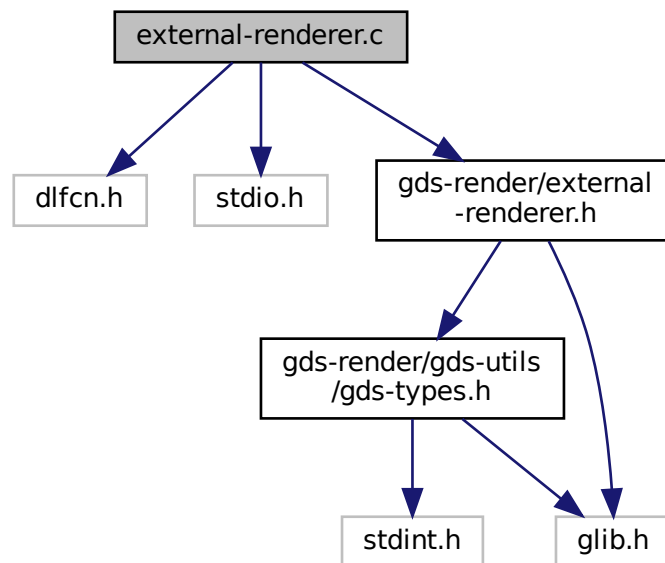
This file implements the dynamic library loading for the external rendering feature.

```

#include <dlfcn.h>
#include <stdio.h>
#include <gds-render/external-renderer.h>

```

Include dependency graph for external-renderer.c:



Functions

- int `external_renderer_render_cell` (struct `gds_cell` *`toplevel_cell`, GList *`layer_info_list`, char *`output_file`, char *`so_path`)
external_renderer_render_cell

13.24.1 Detailed Description

This file implements the dynamic library loading for the external rendering feature.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [external-renderer.c](#).

13.25 external-renderer.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.

```

```

00010 *
00011 * GDSII-Converter is distributed in the hope that it will be useful,
00012 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014 * GNU General Public License for more details.
00015 *
00016 * You should have received a copy of the GNU General Public License
00017 * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018 */
00019
00031 #include <dlfcn.h>
00032 #include <stdio.h>
00033
00034 #include <gds-render/external-renderer.h>
00035
00036 int external_renderer_render_cell(struct gds_cell *toplevel_cell, GList *layer_info_list,
00037     char *output_file, char *so_path)
00038 {
00039     int (*so_render_func)(struct gds_cell *, GList *, char *) = NULL;
00040     void *so_handle = NULL;
00041     char *error_msg;
00042     int ret = 0;
00043
00044     /* Check parameter sanity */
00045     if (!output_file || !so_path || !toplevel_cell || !layer_info_list)
00046         return -3000;
00047
00048     /* Load shared object */
00049     so_handle = dlopen(so_path, RTLD_LAZY);
00050     if (!so_handle) {
00051         printf("Could not load external library '%s'\nDetailed error is:\n%s\n", so_path, dlerror());
00052         return -2000;
00053     }
00054
00055     /* Load symbol from library */
00056     so_render_func = (int (*)(struct gds_cell *, GList *, char *))dlsym(so_handle,
EXTERNAL_LIBRARY_FUNCTION);
00057     error_msg = dlerror();
00058     if (error_msg != NULL) {
00059         printf("Rendering function not found in library:\n%s\n", error_msg);
00060         goto ret_close_so_handle;
00061     }
00062
00063     /* Execute */
00064     if (so_render_func)
00065         so_render_func(toplevel_cell, layer_info_list, output_file);
00066
00067 ret_close_so_handle:
00068     dlclose(so_handle);
00069     return ret;
00070 }
00071

```

13.26 external-renderer.dox File Reference

13.27 external-renderer.h File Reference

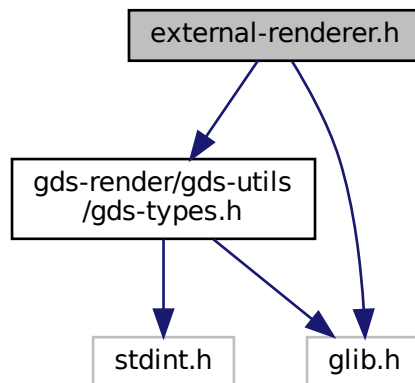
Render according to command line parameters.

```

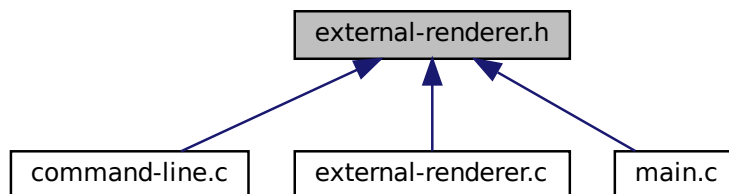
#include <gds-render/gds-utils/gds-types.h>
#include <glib.h>

```

Include dependency graph for external-renderer.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define `EXTERNAL_LIBRARY_FUNCTION` "render_cell_to_file"
function name expected to be found in external library.

Functions

- int `external_renderer_render_cell` (struct `gds_cell` *toplevel_cell, GList *layer_info_list, char *output_file, char *so_path)
external_renderer_render_cell

13.27.1 Detailed Description

Render according to command line parameters.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [external-renderer.h](#).

13.28 external-renderer.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _EXTERNAL_RENDERER_H_
00032 #define _EXTERNAL_RENDERER_H_
00033
00034 #include <gds-render/gds-utils/gds-types.h>
00035 #include <glib.h>
00036
00045 #define EXTERNAL_LIBRARY_FUNCTION "render_cell_to_file"
00046
00055 int external_renderer_render_cell(struct gds_cell *toplevel_cell, GList *layer_info_list, char
    *output_file, char *so_path);
00056
00057 #endif /* _EXTERNAL_RENDERER_H_ */
00058

```

13.29 gds-parser.c File Reference

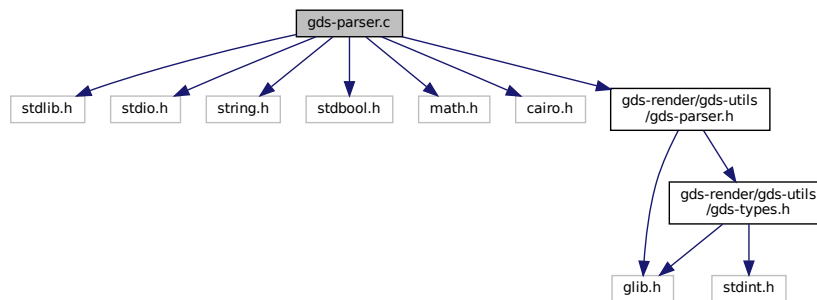
Implementation of the GDS-Parser.

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <cairo.h>

```

```
#include <gds-render/gds-utils/gds-parser.h>
Include dependency graph for gds-parser.c:
```



Macros

- #define `GDS_DEFAULT_UNITS` (10E-9)
Default units assumed for library.
- #define `GDS_ERROR`(fmt, ...) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)
Print GDS error.
- #define `GDS_WARN`(fmt, ...) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)
Print GDS warning.
- #define `GDS_INF`(fmt, ...)

Enumerations

- enum `gds_record` {
`INVALID = 0x0000`, `HEADER = 0x0002`, `BGNLIB = 0x0102`, `LIBNAME = 0x0206`,
`UNITS = 0x0305`, `ENDLIB = 0x0400`, `BGNSTR = 0x0502`, `STRNAME = 0x0606`,
`ENDSTR = 0x0700`, `BOUNDARY = 0x0800`, `PATH = 0x0900`, `SREF = 0x0A00`,
`ENDEL = 0x1100`, `XY = 0x1003`, `MAG = 0x1B05`, `ANGLE = 0x1C05`,
`SNAME = 0x1206`, `STRANS = 0x1A01`, `BOX = 0x2D00`, `LAYER = 0x0D02`,
`WIDTH = 0x0F03`, `PATHTYPE = 0x2102` }

Functions

- static int `name_cell_ref` (struct `gds_cell_instance` *cell_inst, unsigned int bytes, char *data)
Name cell reference.
- static double `gds_convert_double` (const char *data)
Convert GDS 8-byte real to double.
- static signed int `gds_convert_signed_int` (const char *data)
Convert GDS INT32 to int.
- static int16_t `gds_convert_signed_int16` (const char *data)
Convert GDS INT16 to int16.
- static uint16_t `gds_convert_unsigned_int16` (const char *data)
Convert GDS UINT16 String to uint16.
- static GList * `append_library` (GList *curr_list, struct `gds_library` **library_ptr)
Append library to list.

- static GList * [append_graphics](#) (GList *curr_list, enum [graphics_type](#) type, struct [gds_graphics](#) **graphics_ptr)
 - Append graphics to list.*
- static GList * [append_vertex](#) (GList *curr_list, int x, int y)
 - Appends vertex List.*
- static GList * [append_cell](#) (GList *curr_list, struct [gds_cell](#) **cell_ptr)
 - append_cell Append a [gds_cell](#) to a list*
- static GList * [append_cell_ref](#) (GList *curr_list, struct [gds_cell_instance](#) **instance_ptr)
 - Append a cell reference to the reference GList.*
- static int [name_library](#) (struct [gds_library](#) *current_library, unsigned int bytes, char *data)
 - Name a [gds_library](#).*
- static int [name_cell](#) (struct [gds_cell](#) *cell, unsigned int bytes, char *data, struct [gds_library](#) *lib)
 - Names a [gds_cell](#).*
- static void [parse_reference_list](#) (gpointer gcell_ref, gpointer glibrary)
 - Search for cell reference [gcell_ref](#) in [glibrary](#).*
- static void [scan_cell_reference_dependencies](#) (gpointer gcell, gpointer library)
 - Scans cell references inside cell This function searches all the references in [gcell](#) and updates the [gds_cell_instance::cell_ref](#) field in each instance.*
- static void [scan_library_references](#) (gpointer library_list_item, gpointer user)
 - Scans library's cell references.*
- static void [gds_parse_date](#) (const char *buffer, int length, struct [gds_time_field](#) *mod_date, struct [gds_time_field](#) *access_date)
 - [gds_parse_date](#)*
- int [parse_gds_from_file](#) (const char *filename, GList **library_list)
- static void [delete_cell_inst_element](#) (struct [gds_cell_instance](#) *cell_inst)
 - [delete_cell_inst_element](#)*
- static void [delete_vertex](#) (struct [gds_point](#) *vertex)
 - [delete_vertex](#)*
- static void [delete_graphics_obj](#) (struct [gds_graphics](#) *gfx)
 - [delete_graphics_obj](#)*
- static void [delete_cell_element](#) (struct [gds_cell](#) *cell)
 - [delete_cell_element](#)*
- static void [delete_library_element](#) (struct [gds_library](#) *lib)
 - [delete_library_element](#)*
- int [clear_lib_list](#) (GList **library_list)
 - Deletes all libraries including cells, references etc.*

13.29.1 Detailed Description

Implementation of the GDS-Parser.

Author

Mario Hüttel mario.huettel@gmx.net

What's missing? - A lot: Support for 4 Byte real Support for pathtypes Support for datatypes (only layer so far) etc...

Definition in file [gds-parser.c](#).

13.30 gds-parser.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00037 #include <stdlib.h>
00038 #include <stdio.h>
00039 #include <string.h>
00040 #include <stdbool.h>
00041 #include <math.h>
00042 #include <cairo.h>
00043
00044 #include <gds-render/gds-utils/gds-parser.h>
00045
00050 #define GDS_DEFAULT_UNITS (10E-9)
00051
00052 #define GDS_ERROR(fmt, ...) printf("[PARSE_ERROR] " fmt "\n", ##__VA_ARGS__)
00053 #define GDS_WARN(fmt, ...) printf("[PARSE_WARNING] " fmt "\n", ##__VA_ARGS__)
00055 #if GDS_PRINT_DEBUG_INFOS
00056 #define GDS_INF(fmt, ...) printf(fmt, ##__VA_ARGS__)
00057 #else
00058 #define GDS_INF(fmt, ...)
00059 #endif
00060 enum gds_record {
00061     INVALID = 0x0000,
00062     HEADER = 0x0002,
00063     BGNLIB = 0x0102,
00064     LIBNAME = 0x0206,
00065     UNITS = 0x0305,
00066     ENDLIB = 0x0400,
00067     BGNSTR = 0x0502,
00068     STRNAME = 0x0606,
00069     ENDSTR = 0x0700,
00070     BOUNDARY = 0x0800,
00071     PATH = 0x0900,
00072     SREF = 0x0A00,
00073     ENDEL = 0x1100,
00074     XY = 0x1003,
00075     MAG = 0x1B05,
00076     ANGLE = 0x1C05,
00077     SNAME = 0x1206,
00078     STRANS = 0x1A01,
00079     BOX = 0x2D00,
00080     LAYER = 0x0D02,
00081     WIDTH = 0x0F03,
00082     PATHTYPE = 0x2102
00083 };
00084
00092 static int name_cell_ref(struct gds_cell_instance *cell_inst,
00093     unsigned int bytes, char *data)
00094 {
00095     int len;
00096
00097     if (cell_inst == NULL) {
00098         GDS_ERROR("Naming cell ref with no opened cell ref");
00099         return -1;
00100     }
00101     data[bytes] = 0; // Append '0'
00102     len = (int)strlen(data);
00103     if (len > CELL_NAME_MAX-1) {
00104         GDS_ERROR("Cell name '%s' too long: %d\n", data, len);
00105         return -1;
00106     }
00107
00108     /* else: */
00109     strcpy(cell_inst->ref_name, data);
00110     GDS_INF("\tCell referenced: %s\n", cell_inst->ref_name);
00111
00112     return 0;
00113 }
00114

```

```

00120 static double gds_convert_double(const char *data)
00121 {
00122     bool sign_bit;
00123     int i;
00124     double ret_val;
00125     char current_byte;
00126     int bit = 0;
00127     int exponent;
00128
00129     sign_bit = ((data[0] & 0x80) ? true : false);
00130
00131     /* Check for real 0 */
00132     for (i = 0; i < 8; i++) {
00133         if (data[i] != 0)
00134             break;
00135         if (i == 7) {
00136             /* 7 bytes all 0 */
00137             return 0.0;
00138         }
00139     }
00140
00141     /* Value is other than 0 */
00142     ret_val = 0.0;
00143     for (i = 8; i < 64; i++) {
00144         current_byte = data[i/8];
00145         bit = i % 8;
00146         /* isolate bit */
00147         if ((current_byte & (0x80 >> bit)))
00148             ret_val += pow(2, ((double) (-i+7)));
00149     }
00150
00151     /* Parse exponent and sign bit */
00152     exponent = (int)(data[0] & 0x7F);
00153     exponent -= 64;
00154     ret_val *= pow(16, exponent) * (sign_bit == true ? -1 : 1);
00155     return ret_val;
00156 }
00157
00158 }
00159
00165 static signed int gds_convert_signed_int(const char *data)
00166 {
00167     int ret;
00168
00169     if (!data) {
00170         GDS_ERROR("This should not happen");
00171         return 0;
00172     }
00173
00174     ret = (signed int)((((int)data[0] & 0xFF) << 24) |
00175          (((int)data[1] & 0xFF) << 16) |
00176          (((int)(data[2]) & 0xFF) << 8) |
00177          (((int)(data[3]) & 0xFF) << 0));
00178     return ret;
00179 }
00180
00186 static int16_t gds_convert_signed_int16(const char *data)
00187 {
00188     if (!data) {
00189         GDS_ERROR("This should not happen");
00190         return 0;
00191     }
00192     return (int16_t)((((int16_t)(data[0]) & 0xFF) << 8) |
00193                (((int16_t)(data[1]) & 0xFF) << 0));
00194 }
00195
00201 static uint16_t gds_convert_unsigned_int16(const char *data)
00202 {
00203     if (!data) {
00204         GDS_ERROR("This should not happen");
00205         return 0;
00206     }
00207     return (uint16_t)((((uint16_t)(data[0]) & 0xFF) << 8) |
00208                    (((uint16_t)(data[1]) & 0xFF) << 0));
00209 }
00210
00217 static GList *append_library(GList *curr_list, struct gds_library **library_ptr)
00218 {
00219     struct gds_library *lib;
00220
00221     lib = (struct gds_library *)malloc(sizeof(struct gds_library));
00222     if (lib) {
00223         lib->cells = NULL;
00224         lib->name[0] = 0;
00225         lib->unit_in_meters = GDS_DEFAULT_UNITS; // Default. Will be overwritten
00226         lib->cell_names = NULL;
00227     } else

```

```

00228     return NULL;
00229     if (library_ptr)
00230         *library_ptr = lib;
00231
00232     return g_list_append(curr_list, lib);
00233 }
00234
00242 static GList *append_graphics(GList *curr_list, enum graphics_type type,
00243                               struct gds_graphics **graphics_ptr)
00244 {
00245     struct gds_graphics *gfx;
00246
00247     gfx = (struct gds_graphics *)malloc(sizeof(struct gds_graphics));
00248     if (gfx) {
00249         gfx->datatype = 0;
00250         gfx->layer = 0;
00251         gfx->vertices = NULL;
00252         gfx->width_absolute = 0;
00253         gfx->gfx_type = type;
00254         gfx->path_render_type = PATH_FLUSH;
00255     } else
00256         return NULL;
00257
00258     if (graphics_ptr)
00259         *graphics_ptr = gfx;
00260
00261     return g_list_append(curr_list, gfx);
00262 }
00263
00271 static GList *append_vertex(GList *curr_list, int x, int y)
00272 {
00273     struct gds_point *vertex;
00274
00275     vertex = (struct gds_point *)malloc(sizeof(struct gds_point));
00276     if (vertex) {
00277         vertex->x = x;
00278         vertex->y = y;
00279     } else
00280         return NULL;
00281     return g_list_append(curr_list, vertex);
00282 }
00283
00292 static GList *append_cell(GList *curr_list, struct gds_cell **cell_ptr)
00293 {
00294     struct gds_cell *cell;
00295
00296     cell = (struct gds_cell *)malloc(sizeof(struct gds_cell));
00297     if (cell) {
00298         cell->child_cells = NULL;
00299         cell->graphic_objs = NULL;
00300         cell->name[0] = 0;
00301         cell->parent_library = NULL;
00302         cell->checks.unresolved_child_count = GDS_CELL_CHECK_NOT_RUN;
00303         cell->checks.affected_by_reference_loop = GDS_CELL_CHECK_NOT_RUN;
00304     } else
00305         return NULL;
00306     /* return cell */
00307     if (cell_ptr)
00308         *cell_ptr = cell;
00309
00310     return g_list_append(curr_list, cell);
00311 }
00312
00321 static GList *append_cell_ref(GList *curr_list, struct gds_cell_instance **instance_ptr)
00322 {
00323     struct gds_cell_instance *inst;
00324
00325     inst = (struct gds_cell_instance *)
00326           malloc(sizeof(struct gds_cell_instance));
00327     if (inst) {
00328         inst->cell_ref = NULL;
00329         inst->ref_name[0] = 0;
00330         inst->magnification = 1;
00331         inst->flipped = 0;
00332         inst->angle = 0;
00333     } else
00334         return NULL;
00335
00336     if (instance_ptr)
00337         *instance_ptr = inst;
00338
00339     return g_list_append(curr_list, inst);
00340 }
00341
00349 static int name_library(struct gds_library *current_library,
00350                        unsigned int bytes, char *data)
00351 {

```

```

00352     int len;
00353
00354     if (current_library == NULL) {
00355         GDS_ERROR("Naming cell with no opened library");
00356         return -1;
00357     }
00358
00359     data[bytes] = 0; // Append '0'
00360     len = (int)strlen(data);
00361     if (len > CELL_NAME_MAX-1) {
00362         GDS_ERROR("Library name '%s' too long: %d\n", data, len);
00363         return -1;
00364     }
00365
00366     strcpy(current_library->name, data);
00367     GDS_INF("Named library: %s\n", current_library->name);
00368
00369     return 0;
00370 }
00371
00380 static int name_cell(struct gds_cell *cell, unsigned int bytes,
00381                    char *data, struct gds_library *lib)
00382 {
00383     int len;
00384
00385     if (cell == NULL) {
00386         GDS_ERROR("Naming library with no opened library");
00387         return -1;
00388     }
00389     data[bytes] = 0; // Append '0'
00390     len = (int)strlen(data);
00391     if (len > CELL_NAME_MAX-1) {
00392         GDS_ERROR("Cell name '%s' too long: %d\n", data, len);
00393         return -1;
00394     }
00395
00396     strcpy(cell->name, data);
00397     GDS_INF("Named cell: %s\n", cell->name);
00398
00399     /* Append cell name to lib's list of names */
00400     lib->cell_names = g_list_append(lib->cell_names, cell->name);
00401
00402     return 0;
00403 }
00404
00412 static void parse_reference_list(gpointer gcell_ref, gpointer glibrary)
00413 {
00414     struct gds_cell_instance *inst = (struct gds_cell_instance *)gcell_ref;
00415     struct gds_library *lib = (struct gds_library *)glibrary;
00416     GList *cell_item;
00417     struct gds_cell *cell;
00418
00419     GDS_INF("\t\t\tReference: %s: ", inst->ref_name);
00420     /* Find cell */
00421     for (cell_item = lib->cells; cell_item != NULL;
00422          cell_item = cell_item->next) {
00423
00424         cell = (struct gds_cell *)cell_item->data;
00425         /* Check if cell is found */
00426         if (!strcmp(cell->name, inst->ref_name)) {
00427             GDS_INF("found\n");
00428             /* update reference link */
00429             inst->cell_ref = cell;
00430             return;
00431         }
00432     }
00433
00434     GDS_INF("MISSING!\n");
00435     GDS_WARN("referenced cell could not be found in library");
00436 }
00437
00444 static void scan_cell_reference_dependencies(gpointer gcell, gpointer library)
00445 {
00446     struct gds_cell *cell = (struct gds_cell *)gcell;
00447
00448     GDS_INF("\tScanning cell: %s\n", cell->name);
00449
00450     /* Scan all library references */
00451     g_list_foreach(cell->child_cells, parse_reference_list, library);
00452
00453 }
00454
00462 static void scan_library_references(gpointer library_list_item, gpointer user)
00463 {
00464     struct gds_library *lib = (struct gds_library *)library_list_item;
00465     (void)user;
00466

```

```

00467     GDS_INF("Scanning Library: %s\n", lib->name);
00468     g_list_foreach(lib->cells, scan_cell_reference_dependencies, lib);
00469 }
00470
00478 static void gds_parse_date(const char *buffer, int length, struct gds_time_field *mod_date, struct
gds_time_field *access_date)
00479 {
00480     struct gds_time_field *temp_date;
00481     struct gds_time_field *temp_date;
00482
00483     if (!access_date || !mod_date) {
00484         GDS_WARN("Date structures invalid");
00485         return;
00486     }
00487
00488     if (length != (2*6*2)) {
00489         GDS_WARN("Could not parse date field! Not the specified length");
00490         return;
00491     }
00492
00493     for (temp_date = mod_date; 1; temp_date = access_date) {
00494         temp_date->year = gds_convert_unsigend_int16(buffer);
00495         buffer += 2;
00496         temp_date->month = gds_convert_unsigend_int16(buffer);
00497         buffer += 2;
00498         temp_date->day = gds_convert_unsigend_int16(buffer);
00499         buffer += 2;
00500         temp_date->hour = gds_convert_unsigend_int16(buffer);
00501         buffer += 2;
00502         temp_date->minute = gds_convert_unsigend_int16(buffer);
00503         buffer += 2;
00504         temp_date->second = gds_convert_unsigend_int16(buffer);
00505         buffer += 2;
00506
00507         if (temp_date == access_date)
00508             break;
00509     }
00510 }
00511
00512 int parse_gds_from_file(const char *filename, GList **library_list)
00513 {
00514     char *workbuff;
00515     int read;
00516     int i;
00517     int run = 1;
00518     FILE *gds_file = NULL;
00519     uint16_t rec_data_length;
00520     enum gds_record rec_type;
00521     struct gds_library *current_lib = NULL;
00522     struct gds_cell *current_cell = NULL;
00523     struct gds_graphics *current_graphics = NULL;
00524     struct gds_cell_instance *current_s_reference = NULL;
00525     int x, y;
00526     GList *lib_list;
00527
00528     lib_list = *library_list;
00529
00530
00531     /* Allocate working buffer */
00532     workbuff = (char *)malloc(sizeof(char)*128*1024);
00533
00534     if(!workbuff)
00535         return -100;
00536
00537     /* open File */
00538     gds_file = fopen(filename, "rb");
00539     if (gds_file == NULL) {
00540         GDS_ERROR("Could not open File %s", filename);
00541         return -1;
00542     }
00543
00544     /* Record parser */
00545     while (run == 1) {
00546         rec_type = INVALID;
00547         read = fread(workbuff, sizeof(char), 2, gds_file);
00548         if (read != 2 && (current_cell != NULL ||
00549             current_graphics != NULL ||
00550             current_lib != NULL ||
00551             current_s_reference != NULL)) {
00552             GDS_ERROR("End of File. with openend structs/libs");
00553             run = -2;
00554             break;
00555         } else if (read != 2) {
00556             /* EOF */
00557             run = 0;
00558             break;
00559         }
00560

```

```

00561     rec_data_length = gds_convert_unsigend_int16(workbuff);
00562
00563     if (rec_data_length < 4) {
00564         /* Possible Zero-Padding: */
00565         run = 0;
00566         GDS_WARN("Zero Padding detected!");
00567         if (current_cell != NULL ||
00568             current_graphics != NULL ||
00569             current_lib != NULL ||
00570             current_s_reference != NULL) {
00571             GDS_ERROR("Not all structures closed");
00572             run = -2;
00573         }
00574         break;
00575     }
00576     rec_data_length -= 4;
00577
00578     read = fread(workbuff, sizeof(char), 2, gds_file);
00579     if (read != 2) {
00580         run = -2;
00581         GDS_ERROR("Unexpected end of file");
00582         break;
00583     }
00584     rec_type = gds_convert_unsigend_int16(workbuff);
00585
00586     /* if begin: Allocate structures */
00587     switch (rec_type) {
00588     case BGNLIB:
00589         lib_list = append_library(lib_list, &current_lib);
00590         if (lib_list == NULL) {
00591             GDS_ERROR("Allocating memory failed");
00592             run = -3;
00593             break;
00594         }
00595         GDS_INF("Entering Lib\n");
00596         break;
00597     case ENDLIB:
00598         if (current_lib == NULL) {
00599             run = -4;
00600             GDS_ERROR("Closing Library with no opened library");
00601             break;
00602         }
00603     }
00604
00605     /* Check for open Cells */
00606     if (current_cell != NULL) {
00607         run = -4;
00608         GDS_ERROR("Closing Library with opened cells");
00609         break;
00610     }
00611     current_lib = NULL;
00612     GDS_INF("Leaving Library\n");
00613     break;
00614     case BGNSTR:
00615         if (current_lib == NULL) {
00616             GDS_ERROR("Defining Cell outside of library!\n");
00617             run = -4;
00618             break;
00619         }
00620         current_lib->cells = append_cell(current_lib->cells, &current_cell);
00621         if (current_lib->cells == NULL) {
00622             GDS_ERROR("Allocating memory failed");
00623             run = -3;
00624             break;
00625         }
00626
00627         current_cell->parent_library = current_lib;
00628
00629         GDS_INF("Entering Cell\n");
00630         break;
00631     case ENDSTR:
00632         if (current_cell == NULL) {
00633             run = -4;
00634             GDS_ERROR("Closing cell with no opened cell");
00635             break;
00636         }
00637         /* Check for open Elements */
00638         if (current_graphics != NULL || current_s_reference != NULL) {
00639             run = -4;
00640             GDS_ERROR("Closing cell with opened Elements");
00641             break;
00642         }
00643         current_cell = NULL;
00644         GDS_INF("Leaving Cell\n");
00645         break;
00646     case BOX:
00647     case BOUNDARY:

```

```

00648         if (current_cell == NULL) {
00649             GDS_ERROR("Boundary/Box outside of cell");
00650             run = -3;
00651             break;
00652         }
00653         current_cell->graphic_objs = append_graphics(current_cell->graphic_objs,
00654             (rec_type == BOUNDARY ? GRAPHIC_POLYGON : GRAPHIC_BOX),
00655             &current_graphics);
00656         if (current_cell->graphic_objs == NULL) {
00657             GDS_ERROR("Memory allocation failed");
00658             run = -4;
00659             break;
00660         }
00661         GDS_INF("\tEntering boundary/Box\n");
00662         break;
00663     case SREF:
00664         if (current_cell == NULL) {
00665             GDS_ERROR("Cell Reference outside of cell");
00666             run = -3;
00667             break;
00668         }
00669         current_cell->child_cells = append_cell_ref(current_cell->child_cells,
00670             &current_s_reference);
00671         if (current_cell->child_cells == NULL) {
00672             GDS_ERROR("Memory allocation failed");
00673             run = -4;
00674             break;
00675         }
00676
00677         GDS_INF("\tEntering reference\n");
00678         break;
00679     case PATH:
00680         if (current_cell == NULL) {
00681             GDS_ERROR("Path outside of cell");
00682             run = -3;
00683             break;
00684         }
00685         current_cell->graphic_objs = append_graphics(current_cell->graphic_objs,
00686             GRAPHIC_PATH, &current_graphics);
00687         if (current_cell->graphic_objs == NULL) {
00688             GDS_ERROR("Memory allocation failed");
00689             run = -4;
00690             break;
00691         }
00692         GDS_INF("\tEntering Path\n");
00693         break;
00694     case ENDEL:
00695         if (current_graphics != NULL) {
00696
00697             GDS_INF("\tLeaving %s\n", (current_graphics->gfx_type == GRAPHIC_POLYGON ? "boundary"
: "path"));
00698             current_graphics = NULL;
00699         }
00700         if (current_s_reference != NULL) {
00701             GDS_INF("\tLeaving Reference\n");
00702             current_s_reference = NULL;
00703         }
00704         break;
00705     case XY:
00706         if (current_graphics) {
00707
00708         } else if (current_s_reference) {
00709             if (rec_data_length != 8) {
00710                 GDS_WARN("Instance has weird coordinates. Rendered output might be screwed!");
00711             }
00712         }
00713         break;
00714
00715     case MAG:
00716         break;
00717     case ANGLE:
00718         break;
00719     case STRANS:
00720         break;
00721     case WIDTH:
00722         break;
00723     case PATHTYPE:
00724         break;
00725     case UNITS:
00726         break;
00727     default:
00728         //GDS_WARN("Record: %04x, len: %u", (unsigned int)rec_type, (unsigned
int)rec_data_length);
00729         break;
00730     } /* switch(rec_type) */
00731
00732

```



```

00733
00734 /* No Data -> No Processing, go back to top */
00735 if (!rec_data_length || run != 1) continue;
00736
00737 read = fread(workbuff, sizeof(char), rec_data_length, gds_file);
00738
00739 if (read != rec_data_length) {
00740     GDS_ERROR("Could not read enough data: requested: %u, read: %u | Type: 0x%04x\n",
00741             (unsigned int)rec_data_length, (unsigned int)read, (unsigned int)rec_type);
00742     run = -5;
00743     break;
00744 }
00745
00746 switch (rec_type) {
00747
00748     case HEADER:
00749     case ENDLIB:
00750     case ENDSTR:
00751     case BOUNDARY:
00752     case PATH:
00753     case SREF:
00754     case ENDEL:
00755     case BOX:
00756     case INVALID:
00757         break;
00758
00759     case UNITS:
00760         if (!current_lib) {
00761             GDS_WARN("Units defined outside of library!\n");
00762             break;
00763         }
00764
00765         if (rec_data_length != 16) {
00766             GDS_WARN("Unit define incomplete. Will assume database unit of %E meters\n",
current_lib->unit_in_meters);
00767             break;
00768         }
00769
00770         current_lib->unit_in_meters = gds_convert_double(&workbuff[8]);
00771         GDS_INF("Length of database unit: %E meters\n", current_lib->unit_in_meters);
00772         break;
00773     case BGNLIB:
00774         /* Parse date record */
00775         gds_parse_date(workbuff, read, &current_lib->mod_time, &current_lib->access_time);
00776         break;
00777     case BGNSTR:
00778         gds_parse_date(workbuff, read, &current_cell->mod_time, &current_cell->access_time);
00779         break;
00780     case LIBNAME:
00781         name_library(current_lib, (unsigned int)read, workbuff);
00782         break;
00783     case STRNAME:
00784         name_cell(current_cell, (unsigned int)read, workbuff, current_lib);
00785         break;
00786     case XY:
00787         if (current_s_reference) {
00788             /* Get origin of reference */
00789             current_s_reference->origin.x = gds_convert_signed_int(workbuff);
00790             current_s_reference->origin.y = gds_convert_signed_int(&workbuff[4]);
00791             GDS_INF("\t\tSet origin to: %d/%d\n", current_s_reference->origin.x,
00792                 current_s_reference->origin.y);
00793         } else if (current_graphics) {
00794             for (i = 0; i < read/8; i++) {
00795                 x = gds_convert_signed_int(&workbuff[i*8]);
00796                 y = gds_convert_signed_int(&workbuff[i*8+4]);
00797                 current_graphics->vertices =
00798                     append_vertex(current_graphics->vertices, x, y);
00799                 GDS_INF("\t\tSet coordinate: %d/%d\n", x, y);
00800             }
00801         }
00802         break;
00803     case STRANS:
00804         if (!current_s_reference) {
00805             GDS_ERROR("Transformation defined outside of instance");
00806             break;
00807         }
00808         current_s_reference->flipped = ((workbuff[0] & 0x80) ? 1 : 0);
00809         break;
00810     case SNAME:
00811         if (current_s_reference) {
00812             name_cell_ref(current_s_reference, (unsigned int)read, workbuff);
00813         } else {
00814             GDS_ERROR("reference name set outside of cell reference.\n");
00815         }
00816         break;
00817     }
00818 }

```

```

00819     case WIDTH:
00820         if (!current_graphics) {
00821             GDS_WARN("Width defined outside of path element");
00822         }
00823         current_graphics->width_absolute = gds_convert_signed_int(workbuff);
00824         break;
00825     case LAYER:
00826         if (!current_graphics) {
00827             GDS_WARN("Layer has to be defined inside graphics object. Probably unknown object.
Implement it yourself!");
00828             break;
00829         }
00830         current_graphics->layer = gds_convert_signed_int16(workbuff);
00831         if (current_graphics->layer < 0) {
00832             GDS_WARN("Layer negative!\n");
00833         }
00834         GDS_INF("\t\tAdded layer %d\n", (int)current_graphics->layer);
00835         break;
00836     case MAG:
00837         if (rec_data_length != 8) {
00838             GDS_WARN("Magnification is not an 8 byte real. Results may be wrong");
00839         }
00840         if (current_graphics != NULL && current_s_reference != NULL) {
00841             GDS_ERROR("Open Graphics and Cell Reference\n\tMissing ENDEL?");
00842             run = -6;
00843             break;
00844         }
00845         if (current_s_reference != NULL) {
00846             current_s_reference->magnification = gds_convert_double(workbuff);
00847             GDS_INF("\t\tMagnification defined: %lf\n", current_s_reference->magnification);
00848         }
00849         break;
00850     case ANGLE:
00851         if (rec_data_length != 8) {
00852             GDS_WARN("Angle is not an 8 byte real. Results may be wrong");
00853         }
00854         if (current_graphics != NULL && current_s_reference != NULL) {
00855             GDS_ERROR("Open Graphics and Cell Reference\n\tMissing ENDEL?");
00856             run = -6;
00857             break;
00858         }
00859         if (current_s_reference != NULL) {
00860             current_s_reference->angle = gds_convert_double(workbuff);
00861             GDS_INF("\t\tAngle defined: %lf\n", current_s_reference->angle);
00862         }
00863         break;
00864     case PATHTYPE:
00865         if (current_graphics == NULL) {
00866             GDS_WARN("Path type defined outside of path. Ignoring");
00867             break;
00868         }
00869         if (current_graphics->gfx_type == GRAPHIC_PATH) {
00870             current_graphics->path_render_type = (enum
path_type)gds_convert_signed_int16(workbuff);
00871             GDS_INF("\t\tPathtype: %d\n", current_graphics->path_render_type);
00872         } else {
00873             GDS_WARN("Path type defined inside non-path graphics object. Ignoring");
00874         }
00875         break;
00876     }
00877 }
00878
00879 } /* while(run == 1) */
00880
00881 fclose(gds_file);
00882
00883 if (!run) {
00884     /* Iterate and find references to cells */
00885     g_list_foreach(lib_list, scan_library_references, NULL);
00886 }
00887
00888 *library_list = lib_list;
00889
00890 free(workbuff);
00891
00892 return run;
00893 }
00894
00899 static void delete_cell_inst_element(struct gds_cell_instance *cell_inst)
00900 {
00901     if (cell_inst)
00902         free(cell_inst);
00903 }
00904
00909 static void delete_vertex(struct gds_point *vertex)
00910 {
00911     if (vertex)

```

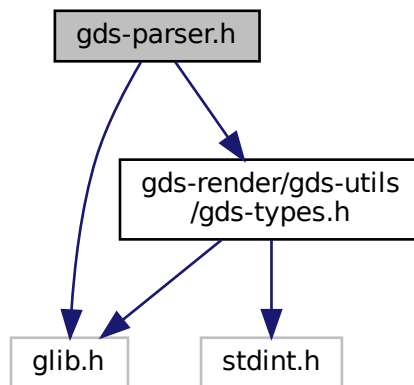
```
00912         free(vertex);
00913     }
00914
00919 static void delete_graphics_obj(struct gds_graphics *gfx)
00920 {
00921     if (!gfx)
00922         return;
00923
00924     g_list_free_full(gfx->vertices, (GDestroyNotify)delete_vertex);
00925     free(gfx);
00926 }
00927
00932 static void delete_cell_element(struct gds_cell *cell)
00933 {
00934     if (!cell)
00935         return;
00936
00937     g_list_free_full(cell->child_cells, (GDestroyNotify)delete_cell_inst_element);
00938     g_list_free_full(cell->graphic_objs, (GDestroyNotify)delete_graphics_obj);
00939     free(cell);
00940 }
00941
00946 static void delete_library_element(struct gds_library *lib)
00947 {
00948     if (!lib)
00949         return;
00950
00951     g_list_free(lib->cell_names);
00952     g_list_free_full(lib->cells, (GDestroyNotify)delete_cell_element);
00953     free(lib);
00954 }
00955
00956 int clear_lib_list(GList **library_list)
00957 {
00958     if (!library_list)
00959         return 0;
00960
00961     if (*library_list == NULL)
00962         return 0;
00963
00964     g_list_free_full(*library_list, (GDestroyNotify)delete_library_element);
00965     *library_list = NULL;
00966     return 0;
00967 }
00968
```

13.31 gds-parser.h File Reference

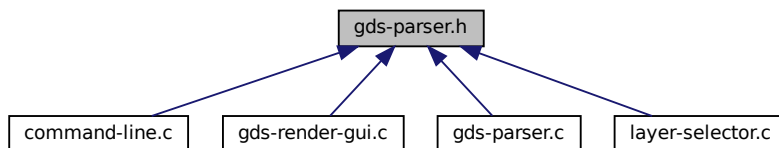
Header file for the GDS-Parser.

```
#include <glib.h>
#include <gds-render/gds-utils/gds-types.h>
```

Include dependency graph for `gds-parser.h`:



This graph shows which files directly or indirectly include this file:



Macros

- `#define GDS_PRINT_DEBUG_INFOS (0)`
1: Print infos, 0: Don't print

Functions

- `int parse_gds_from_file (const char *filename, GList **library_list)`
- `int clear_lib_list (GList **library_list)`
Deletes all libraries including cells, references etc.

13.31.1 Detailed Description

Header file for the GDS-Parser.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [gds-parser.h](#).

13.32 gds-parser.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _GDSPARSER_H_
00032 #define _GDSPARSER_H_
00033
00034 #include <glib.h>
00035
00036 #include <gds-render/gds-utils/gds-types.h>
00037
00038 #define GDS_PRINT_DEBUG_INFOS (0)
00040 int parse_gds_from_file(const char *filename, GList **library_array);
00041
00046 int clear_lib_list(GList **library_list);
00047
00050 #endif /* _GDSPARSE_H_ */

```

13.33 gds-render-gui.c File Reference

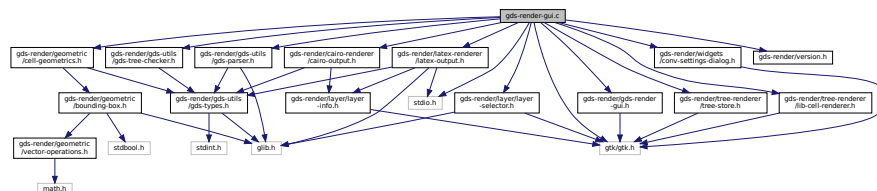
Handling of GUI.

```

#include <stdio.h>
#include <gtk/gtk.h>
#include <gds-render/gds-render-gui.h>
#include <gds-render/gds-utils/gds-parser.h>
#include <gds-render/gds-utils/gds-tree-checker.h>
#include <gds-render/layer/layer-selector.h>
#include <gds-render/tree-renderer/tree-store.h>
#include <gds-render/tree-renderer/lib-cell-renderer.h>
#include <gds-render/latex-renderer/latex-output.h>
#include <gds-render/cairo-renderer/cairo-output.h>
#include <gds-render/widgets/conv-settings-dialog.h>
#include <gds-render/geometric/cell-geometrics.h>
#include <gds-render/version.h>

```

Include dependency graph for gds-render-gui.c:



Data Structures

- struct [_GdsRenderGui](#)

Enumerations

- enum `gds_render_gui_signal_sig_ids` { `SIGNAL_WINDOW_CLOSED` = 0, `SIGNAL_COUNT` }

Functions

- static gboolean `on_window_close` (gpointer window, GdkEvent *event, gpointer user)
Main window close event.
- static GString * `generate_string_from_date` (struct `gds_time_field` *date)
generate string from `gds_time_field`
- static void `on_load_gds` (gpointer button, gpointer user)
Callback function of Load GDS button.
- static void `on_convert_clicked` (gpointer button, gpointer user)
Convert button callback.
- static void `cell_tree_view_activated` (gpointer tree_view, GtkTreePath *path, GtkTreeViewColumn *column, gpointer user)
cell_tree_view_activated Callback for 'double click' on cell selector element
- static void `cell_selection_changed` (GtkTreeSelection *sel, GdsRenderGui *self)
Callback for cell-selection change event.
- static void `sort_up_callback` (GtkWidget *widget, gpointer user)
- static void `sort_down_callback` (GtkWidget *widget, gpointer user)
- static void `gds_render_gui_dispose` (GObject *gobject)
- static void `gds_render_gui_class_init` (GdsRenderGuiClass *klass)
- GtkWidget * `gds_render_gui_get_main_window` (GdsRenderGui *gui)
Get main window.
- static void `gds_render_gui_init` (GdsRenderGui *self)
- GdsRenderGui * `gds_render_gui_new` ()
Create new GdsRenderGui Object.

Variables

- static guint `gds_render_gui_signals` [`SIGNAL_COUNT`]

13.33.1 Detailed Description

Handling of GUI.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [gds-render-gui.c](#).

13.34 gds-render-gui.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00030 #include <stdio.h>
00031 #include <gtk/gtk.h>
00032
00033 #include <gds-render/gds-render-gui.h>
00034 #include <gds-render/gds-utils/gds-parser.h>
00035 #include <gds-render/gds-utils/gds-tree-checker.h>
00036 #include <gds-render/layer/layer-selector.h>
00037 #include <gds-render/tree-renderer/tree-store.h>
00038 #include <gds-render/tree-renderer/lib-cell-renderer.h>
00039 #include <gds-render/latex-renderer/latex-output.h>
00040 #include <gds-render/cairo-renderer/cairo-output.h>
00041 #include <gds-render/widgets/conv-settings-dialog.h>
00042 #include <gds-render/geometric/cell-geometrics.h>
00043 #include <gds-render/version.h>
00044
00045 enum gds_render_gui_signal_sig_ids {SIGNAL_WINDOW_CLOSED = 0, SIGNAL_COUNT};
00046
00047 static guint gds_render_gui_signals[SIGNAL_COUNT];
00048
00049 struct _GdsRenderGui {
00050     /* Parent GObject */
00051     GObject parent;
00052
00053     /* Custom fields */
00054     GtkWidget *main_window;
00055     GtkWidget *convert_button;
00056     GtkTreeStore *cell_tree_store;
00057     GtkWidget *cell_search_entry;
00058     LayerSelector *layer_selector;
00059     GtkTreeView *cell_tree_view;
00060     GList *gds_libraries;
00061     struct render_settings render_dialog_settings;
00062 };
00063
00064 G_DEFINE_TYPE(GdsRenderGui, gds_render_gui, G_TYPE_OBJECT)
00065
00066
00073 static gboolean on_window_close(gpointer window, GdkEvent *event, gpointer user)
00074 {
00075     GdsRenderGui *self;
00076
00077     self = RENDERER_GUI(user);
00078     /* Don't close window in case of error */
00079     if (!self)
00080         return TRUE;
00081
00082     /* Close Window. Leads to termination of the program/the current instance */
00083     g_clear_object(&self->main_window);
00084     gtk_widget_destroy(GTK_WIDGET(window));
00085
00086     /* Delete loaded library data */
00087     clear_lib_list(&self->gds_libraries);
00088
00089     g_signal_emit(self, gds_render_gui_signals[SIGNAL_WINDOW_CLOSED], 0);
00090
00091     return TRUE;
00092 }
00093
00099 static GString *generate_string_from_date(struct gds_time_field *date)
00100 {
00101     GString *str;
00102
00103     str = g_string_new_len(NULL, 50);
00104     g_string_printf(str, "%02u.%02u.%u - %02u:%02u",
00105                   (unsigned int)date->day,
00106                   (unsigned int)date->month,

```

```

00107         (unsigned int)date->year,
00108         (unsigned int)date->hour,
00109         (unsigned int)date->minute);
00110     return str;
00111 }
00112
00118 static void on_load_gds(gpointer button, gpointer user)
00119 {
00120     GList *cell;
00121     GtkTreeIter libiter;
00122     GtkTreeIter celliter;
00123     GList *lib;
00124     struct gds_library *gds_lib;
00125     struct gds_cell *gds_c;
00126     GdsRenderGui *self;
00127     GtkWidget *open_dialog;
00128     GtkFileChooser *file_chooser;
00129     GtkFileFilter *filter;
00130     GtkStyleContext *button_style;
00131     gint dialog_result;
00132     int gds_result;
00133     char *filename;
00134     GString *mod_date;
00135     GString *acc_date;
00136     unsigned int cell_error_level;
00137
00138     self = RENDERER_GUI(user);
00139     if (!self)
00140         return;
00141
00142     open_dialog = gtk_file_chooser_dialog_new("Open GDSII File", self->main_window,
00143                                             GTK_FILE_CHOOSER_ACTION_OPEN,
00144                                             "Cancel", GTK_RESPONSE_CANCEL,
00145                                             "Open GDSII", GTK_RESPONSE_ACCEPT,
00146                                             NULL);
00147     file_chooser = GTK_FILE_CHOOSER(open_dialog);
00148
00149     /* Add GDS II Filter */
00150     filter = gtk_file_filter_new();
00151     gtk_file_filter_add_pattern(filter, "*.gds");
00152     gtk_file_filter_set_name(filter, "GDSII-Files");
00153     gtk_file_chooser_add_filter(file_chooser, filter);
00154
00155     dialog_result = gtk_dialog_run(GTK_DIALOG(open_dialog));
00156
00157     if (dialog_result != GTK_RESPONSE_ACCEPT)
00158         goto end_destroy;
00159
00160     /* Get File name */
00161     filename = gtk_file_chooser_get_filename(file_chooser);
00162
00163     gtk_tree_store_clear(self->cell_tree_store);
00164     clear_lib_list(&self->gds_libraries);
00165
00166     /* Parse new GDSII file */
00167     gds_result = parse_gds_from_file(filename, &self->gds_libraries);
00168
00169     /* Delete file name afterwards */
00170     g_free(filename);
00171     if (gds_result)
00172         goto end_destroy;
00173
00174     /* remove suggested action from Open button */
00175     button_style = gtk_widget_get_style_context(GTK_WIDGET(button));
00176     gtk_style_context_remove_class(button_style, "suggested-action");
00177
00178     for (lib = self->gds_libraries; lib != NULL; lib = lib->next) {
00179         gds_lib = (struct gds_library *)lib->data;
00180         /* Create top level iter */
00181         gtk_tree_store_append(self->cell_tree_store, &libiter, NULL);
00182
00183         /* Convert dates to String */
00184         mod_date = generate_string_from_date(&gds_lib->mod_time);
00185         acc_date = generate_string_from_date(&gds_lib->access_time);
00186
00187         gtk_tree_store_set(self->cell_tree_store, &libiter,
00188                           CELL_SEL_LIBRARY, gds_lib,
00189                           CELL_SEL_MODDATE, mod_date->str,
00190                           CELL_SEL_ACCESSDATE, acc_date->str,
00191                           -1);
00192
00193         /* Check this library. This might take a while */
00194         (void)gds_tree_check_cell_references(gds_lib);
00195         (void)gds_tree_check_reference_loops(gds_lib);
00196         /* Delete GStrings including string data. */
00197         /* Cell store copies String type data items */
00198         g_string_free(mod_date, TRUE);

```



```

00199     g_string_free(acc_date, TRUE);
00200
00201     for (cell = gds_lib->cells; cell != NULL; cell = cell->next) {
00202         gds_c = (struct gds_cell *)cell->data;
00203         gtk_tree_store_append(self->cell_tree_store, &celliter, &libiter);
00204         /* Convert dates to String */
00205         mod_date = generate_string_from_date(&gds_c->mod_time);
00206         acc_date = generate_string_from_date(&gds_c->access_time);
00207
00208         /* Get the checking results for this cell */
00209         cell_error_level = 0;
00210         if (gds_c->checks.unresolved_child_count)
00211             cell_error_level |= LIB_CELL_RENDERER_ERROR_WARN;
00212
00213         /* Check if it is completely broken */
00214         if (gds_c->checks.affected_by_reference_loop)
00215             cell_error_level |= LIB_CELL_RENDERER_ERROR_ERR;
00216
00217         /* Add cell to tree store model */
00218         gtk_tree_store_set(self->cell_tree_store, &celliter,
00219             CELL_SEL_CELL, gds_c,
00220             CELL_SEL_MODDATE, mod_date->str,
00221             CELL_SEL_ACCESSDATE, acc_date->str,
00222             CELL_SEL_CELL_ERROR_STATE, cell_error_level,
00223             -1);
00224
00225         /* Delete GStrings including string data. */
00226         /* Cell store copies String type data items */
00227         g_string_free(mod_date, TRUE);
00228         g_string_free(acc_date, TRUE);
00229     } /* for cells */
00230 } /* for libraries */
00231
00232 /* Create Layers in Layer Box */
00233 layer_selector_generate_layer_widgets(self->layer_selector, self->gds_libraries);
00234
00235 end_destroy:
00236 /* Destroy dialog and filter */
00237 gtk_widget_destroy(open_dialog);
00238 }
00239
00240 static void on_convert_clicked(gpointer button, gpointer user)
00241 {
00242     (void)button;
00243     GdsRenderGui *self;
00244     GtkTreeSelection *selection;
00245     GtkTreeIter iter;
00246     GtkTreeModel *model;
00247     GList *layer_list;
00248     struct gds_cell *cell_to_render;
00249     FILE *output_file;
00250     GtkWidget *dialog;
00251     RendererSettingsDialog *settings;
00252     GtkFileFilter *filter;
00253     gint res;
00254     char *file_name;
00255     union bounding_box cell_box;
00256     unsigned int height, width;
00257     struct render_settings *sett;
00258
00259     self = RENDERER_GUI(user);
00260
00261     if (!self)
00262         return;
00263
00264     sett = &self->render_dialog_settings;
00265
00266     /* Get selected cell */
00267     selection = gtk_tree_view_get_selection(self->cell_tree_view);
00268     if (gtk_tree_selection_get_selected(selection, &model, &iter) == FALSE)
00269         return;
00270
00271     gtk_tree_model_get(model, &iter, CELL_SEL_CELL, &cell_to_render, -1);
00272
00273     if (!cell_to_render)
00274         return;
00275
00276     /* Get layers that are rendered */
00277     layer_list = layer_selector_export_rendered_layer_info(self->layer_selector);
00278
00279     /* Calculate cell size in DB units */
00280     bounding_box_prepare_empty(&cell_box);
00281     calculate_cell_bounding_box(&cell_box, cell_to_render);
00282
00283     /* Calculate size in database units
00284     * Note that the results are bound to be positive,
00285     * so casting them to unsigned int is absolutely valid
00286     */

```

```

00291     */
00292     height = (unsigned int)(cell_box.vectors.upper_right.y - cell_box.vectors.lower_left.y);
00293     width = (unsigned int)(cell_box.vectors.upper_right.x - cell_box.vectors.lower_left.x);
00294
00295     /* Show settings dialog */
00296     settings = renderer_settings_dialog_new(GTK_WINDOW(self->main_window));
00297     renderer_settings_dialog_set_settings(settings, sett);
00298     renderer_settings_dialog_set_database_unit_scale(settings,
cell_to_render->parent_library->unit_in_meters);
00299     renderer_settings_dialog_set_cell_height(settings, height);
00300     renderer_settings_dialog_set_cell_width(settings, width);
00301     g_object_set(G_OBJECT(settings), "cell-name", cell_to_render->name, NULL);
00302
00303     res = gtk_dialog_run(GTK_DIALOG(settings));
00304     if (res == GTK_RESPONSE_OK) {
00305         renderer_settings_dialog_get_settings(settings, sett);
00306         gtk_widget_destroy(GTK_WIDGET(settings));
00307     } else {
00308         gtk_widget_destroy(GTK_WIDGET(settings));
00309         goto ret_layer_destroy;
00310     }
00311
00312     /* save file dialog */
00313     dialog = gtk_file_chooser_dialog_new((sett->renderer == RENDERER_LATEX_TIKZ
00314         ? "Save LaTeX File" : "Save PDF"),
00315         GTK_WINDOW(self->main_window), GTK_FILE_CHOOSER_ACTION_SAVE,
00316         "Cancel", GTK_RESPONSE_CANCEL, "Save", GTK_RESPONSE_ACCEPT, NULL);
00317     /* Set file filter according to settings */
00318     filter = gtk_file_filter_new();
00319     switch (sett->renderer) {
00320     case RENDERER_LATEX_TIKZ:
00321         gtk_file_filter_add_pattern(filter, "*.tex");
00322         gtk_file_filter_set_name(filter, "LaTeX-Files");
00323         break;
00324     case RENDERER_CAIROGRAPHICS_PDF:
00325         gtk_file_filter_add_pattern(filter, "*.pdf");
00326         gtk_file_filter_set_name(filter, "PDF-Files");
00327         break;
00328     case RENDERER_CAIROGRAPHICS_SVG:
00329         gtk_file_filter_add_pattern(filter, "*.svg");
00330         gtk_file_filter_set_name(filter, "SVG-Files");
00331         break;
00332     }
00333
00334     gtk_file_chooser_add_filter(GTK_FILE_CHOOSER(dialog), filter);
00335
00336     gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
00337
00338     res = gtk_dialog_run(GTK_DIALOG(dialog));
00339     if (res == GTK_RESPONSE_ACCEPT) {
00340         file_name = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
00341         gtk_widget_destroy(dialog);
00342
00343         switch (sett->renderer) {
00344         case RENDERER_LATEX_TIKZ:
00345             output_file = fopen(file_name, "w");
00346             latex_render_cell_to_code(cell_to_render, layer_list, output_file, sett->scale,
00347                 sett->tex_pdf_layers, sett->tex_standalone);
00348             fclose(output_file);
00349             break;
00350         case RENDERER_CAIROGRAPHICS_SVG:
00351         case RENDERER_CAIROGRAPHICS_PDF:
00352             cairo_render_cell_to_vector_file(cell_to_render, layer_list,
00353                 (sett->renderer == RENDERER_CAIROGRAPHICS_PDF
00354                 ? file_name
00355                 : NULL),
00356                 (sett->renderer == RENDERER_CAIROGRAPHICS_SVG
00357                 ? file_name
00358                 : NULL),
00359                 sett->scale);
00360             break;
00361         }
00362         g_free(file_name);
00363     } else {
00364         gtk_widget_destroy(dialog);
00365     }
00366
00367 ret_layer_destroy:
00368     g_list_free_full(layer_list, (GDestroyNotify)layer_info_delete_struct);
00369 }
00370
00371 static void cell_tree_view_activated(gpointer tree_view, GtkTreePath *path,
00372     GtkTreeViewColumn *column, gpointer user)
00373 {
00374     (void)tree_view;
00375     (void)path;
00376     (void)column;

```

```

00384
00385     on_convert_clicked(NULL, user);
00386 }
00387
00388
00397 static void cell_selection_changed(GtkTreeSelection *sel, GdsRenderGui *self)
00398 {
00399     GtkTreeModel *model = NULL;
00400     GtkTreeIter iter;
00401
00402     if (gtk_tree_selection_get_selected(sel, &model, &iter)) {
00403         /* Node selected. Show button */
00404         gtk_widget_set_sensitive(self->convert_button, TRUE);
00405     } else {
00406         gtk_widget_set_sensitive(self->convert_button, FALSE);
00407     }
00408 }
00409
00410 static void sort_up_callback(GtkWidget *widget, gpointer user)
00411 {
00412     (void)widget;
00413     GdsRenderGui *self;
00414
00415     self = RENDERER_GUI(user);
00416     if (!self)
00417         return;
00418     layer_selector_force_sort(self->layer_selector, LAYER_SELECTOR_SORT_UP);
00419 }
00420
00421 static void sort_down_callback(GtkWidget *widget, gpointer user)
00422 {
00423     (void)widget;
00424     GdsRenderGui *self;
00425
00426     self = RENDERER_GUI(user);
00427     if (!self)
00428         return;
00429     layer_selector_force_sort(self->layer_selector, LAYER_SELECTOR_SORT_DOWN);
00430 }
00431
00432 static void gds_render_gui_dispose(GObject *gobject)
00433 {
00434     GdsRenderGui *self;
00435
00436     self = RENDERER_GUI(gobject);
00437
00438     clear_lib_list(&self->gds_libraries);
00439
00440     g_clear_object(&self->cell_tree_view);
00441     g_clear_object(&self->convert_button);
00442     g_clear_object(&self->layer_selector);
00443     g_clear_object(&self->cell_tree_store);
00444     g_clear_object(&self->cell_search_entry);
00445
00446     if (self->main_window) {
00447         g_signal_handlers_destroy(self->main_window);
00448         gtk_widget_destroy(GTK_WIDGET(self->main_window));
00449         self->main_window = NULL;
00450     }
00451
00452     /* Chain up */
00453     G_OBJECT_CLASS(gds_render_gui_parent_class)->dispose(gobject);
00454 }
00455
00456 static void gds_render_gui_class_init(GdsRenderGuiClass *klass)
00457 {
00458     GObjectClass *gobject_class = G_OBJECT_CLASS(klass);
00459
00460     gds_render_gui_signals[SIGNAL_WINDOW_CLOSED] =
00461         g_signal_newv("window-closed", RENDERER_TYPE_GUI,
00462                     G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE,
00463                     NULL,
00464                     NULL,
00465                     NULL,
00466                     NULL,
00467                     G_TYPE_NONE,
00468                     0,
00469                     NULL);
00470
00471     gobject_class->dispose = gds_render_gui_dispose;
00472 }
00473
00474 GtkWidget *gds_render_gui_get_main_window(GdsRenderGui *gui)
00475 {
00476     return gui->main_window;
00477 }
00478

```

```

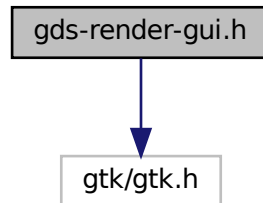
00479 static void gds_render_gui_init(GdsRenderGui *self)
00480 {
00481     GtkBuilder *main_builder;
00482     GtkWidget *listbox;
00483     GtkHeaderBar *header_bar;
00484     struct tree_stores *cell_selector_stores;
00485     GtkWidget *sort_up_button;
00486     GtkWidget *sort_down_button;
00487
00488     main_builder = gtk_builder_new_from_resource("/main.glade");
00489
00490     self->cell_tree_view = GTK_TREE_VIEW(gtk_builder_get_object(main_builder, "cell-tree"));
00491     self->cell_search_entry = GTK_WIDGET(gtk_builder_get_object(main_builder, "cell-search"));
00492
00493     cell_selector_stores = setup_cell_selector(self->cell_tree_view,
GTK_ENTRY(self->cell_search_entry));
00494
00495     self->cell_tree_store = cell_selector_stores->base_store;
00496
00497     self->main_window = GTK_WINDOW(gtk_builder_get_object(main_builder, "main-window"));
00498     g_signal_connect(GTK_WIDGET(gtk_builder_get_object(main_builder, "button-load-gds")),
00499                     "clicked", G_CALLBACK(on_load_gds), (gpointer)self);
00500
00501     self->convert_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "convert-button"));
00502     g_signal_connect(self->convert_button, "clicked", G_CALLBACK(on_convert_clicked), (gpointer)self);
00503
00504     listbox = GTK_WIDGET(gtk_builder_get_object(main_builder, "layer-list"));
00505     /* Create layer selector */
00506     self->layer_selector = layer_selector_new(GTK_LIST_BOX(listbox));
00507
00508
00509     /* Callback for selection change of cell selector */
00510     g_signal_connect(G_OBJECT(gtk_tree_view_get_selection(self->cell_tree_view)), "changed",
00511                     G_CALLBACK(cell_selection_changed), self);
00512     g_signal_connect(self->cell_tree_view, "row-activated", G_CALLBACK(cell_tree_view_activated),
self);
00513
00514     /* Set version in main window subtitle */
00515     header_bar = GTK_HEADER_BAR(gtk_builder_get_object(main_builder, "header-bar"));
00516     gtk_header_bar_set_subtitle(header_bar, _app_version_string);
00517
00518     /* Get layer sorting buttons and set callbacks */
00519     sort_up_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "button-up-sort"));
00520     sort_down_button = GTK_WIDGET(gtk_builder_get_object(main_builder, "button-down-sort"));
00521
00522     g_signal_connect(sort_up_button, "clicked", G_CALLBACK(sort_up_callback), self);
00523     g_signal_connect(sort_down_button, "clicked", G_CALLBACK(sort_down_callback), self);
00524
00525     /* Set buttons for loading and saving */
00526     layer_selector_set_load_mapping_button(self->layer_selector,
00527                                           GTK_WIDGET(gtk_builder_get_object(main_builder, "button-load-mapping")),
00528                                           self->main_window);
00529     layer_selector_set_save_mapping_button(self->layer_selector,
GTK_WIDGET(gtk_builder_get_object(main_builder, "button-save-mapping")),
self->main_window);
00530
00531
00532     /* Connect delete-event */
00533     g_signal_connect(GTK_WIDGET(self->main_window), "delete-event",
00534                     G_CALLBACK(on_window_close), self);
00535
00536     g_object_unref(main_builder);
00537
00538     /* Set default conversion/rendering settings */
00539     self->render_dialog_settings.scale = 1000;
00540     self->render_dialog_settings.renderer = RENDERER_LATEX_TIKZ;
00541     self->render_dialog_settings.tex_pdf_layers = FALSE;
00542     self->render_dialog_settings.tex_standalone = FALSE;
00543
00544
00545     /* Reference all objects referenced by this object */
00546     g_object_ref(self->main_window);
00547     g_object_ref(self->cell_tree_view);
00548     g_object_ref(self->convert_button);
00549     g_object_ref(self->layer_selector);
00550     g_object_ref(self->cell_tree_store);
00551     g_object_ref(self->cell_search_entry);
00552 }
00553
00554 GdsRenderGui *gds_render_gui_new()
00555 {
00556     return RENDERER_GUI(g_object_new(RENDERER_TYPE_GUI, NULL));
00557 }
00558

```

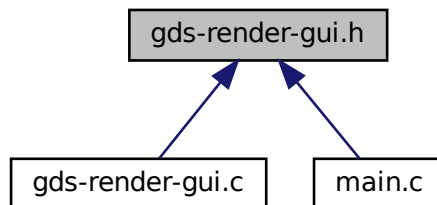
13.35 gds-render-gui.h File Reference

Header for GdsRenderGui Object.

```
#include <gtk/gtk.h>  
Include dependency graph for gds-render-gui.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define [RENDERER_TYPE_GUI](#) (gds_render_gui_get_type())

Functions

- G_BEGIN_DECLS [G_DECLARE_FINAL_TYPE](#) (GdsRenderGui, gds_render_gui, RENDERER, GUI, G↔Object)
- GdsRenderGui * [gds_render_gui_new](#) ()
Create new GdsRenderGui Object.
- GtkWidget * [gds_render_gui_get_main_window](#) (GdsRenderGui *gui)
Get main window.

13.35.1 Detailed Description

Header for GdsRenderGui Object.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [gds-render-gui.h](#).

13.36 gds-render-gui.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef _GDS_RENDER_GUI_
00027 #define _GDS_RENDER_GUI_
00028
00034 #include <gtk/gtk.h>
00035
00036 G_BEGIN_DECLS
00037
00038 G_DECLARE_FINAL_TYPE(GdsRenderGui, gds_render_gui, RENDERER, GUI, GObject);
00039
00040 #define RENDERER_TYPE_GUI (gds_render_gui_get_type())
00041
00046 GdsRenderGui *gds_render_gui_new();
00047
00055 GtkWidget *gds_render_gui_get_main_window(GdsRenderGui *gui);
00056
00057 G_END_DECLS
00058
00061 #endif /* _GDS_RENDER_GUI_ */

```

13.37 gds-tree-checker.c File Reference

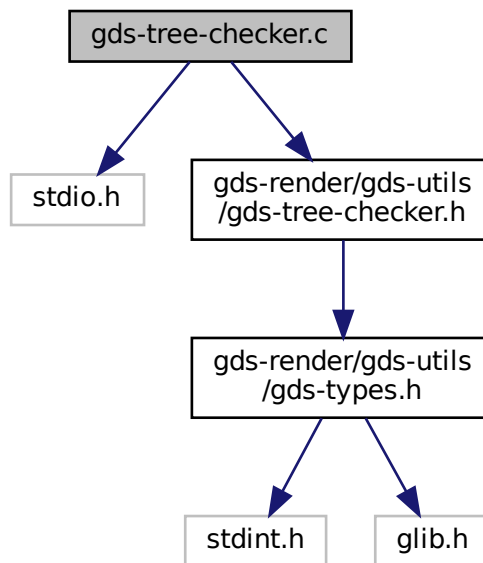
Checking functions of a cell tree.

```

#include <stdio.h>
#include <gds-render/gds-utils/gds-tree-checker.h>

```

Include dependency graph for gds-tree-checker.c:



Functions

- int [gds_tree_check_cell_references](#) (struct [gds_library](#) *lib)
gds_tree_check_cell_references checks if all child cell references can be resolved in the given library
- static int [gds_tree_check_list_contains_cell](#) (GList *list, struct [gds_cell](#) *cell)
Check if list contains a cell.
- static int [gds_tree_check_iterate_ref_and_check](#) (struct [gds_cell](#) *cell_to_check, GList **visited_cells)
This function follows down the reference list of a cell and marks each visited subcell and detects loops.
- int [gds_tree_check_reference_loops](#) (struct [gds_library](#) *lib)
gds_tree_check_reference_loops checks if the given library contains reference loops

13.37.1 Detailed Description

Checking functions of a cell tree.

This file contains checking functions for the GDS cell tree. These functions include checks if all child references could be resolved, and if the cell tree contains loops.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [gds-tree-checker.c](#).

13.38 gds-tree-checker.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00036 #include <stdio.h>
00037
00038 #include <gds-render/gds-utils/gds-tree-checker.h>
00039
00040 int gds_tree_check_cell_references(struct gds_library *lib)
00041 {
00042     GList *cell_iter;
00043     struct gds_cell *cell;
00044     GList *instance_iter;
00045     struct gds_cell_instance *cell_inst;
00046     int total_unresolved_count = 0;
00047
00048     if (!lib)
00049         return -1;
00050
00051     /* Iterate over all cells in library */
00052     for (cell_iter = lib->cells; cell_iter != NULL; cell_iter = g_list_next(cell_iter)) {
00053         cell = (struct gds_cell *)cell_iter->data;
00054
00055         /* Check if this list element is broken. This should never happen */
00056         if (!cell) {
00057             fprintf(stderr, "Broken cell list item found. Will continue.\n");
00058             continue;
00059         }
00060
00061         /* Reset the unresolved cell reference counter to 0 */
00062         cell->checks.unresolved_child_count = 0;
00063
00064         /* Iterate through all child cell references and check if the references are set */
00065         for (instance_iter = cell->child_cells; instance_iter != NULL;
00066             instance_iter = g_list_next(instance_iter)) {
00067             cell_inst = (struct gds_cell_instance *)instance_iter->data;
00068
00069             /* Check if broken. This should not happen */
00070             if (!cell_inst) {
00071                 fprintf(stderr, "Broken cell list item found in cell %s. Will continue.\n",
00072                     cell->name);
00073                 continue;
00074             }
00075
00076             /* Check if instance is valid; else increment "error" counter of cell */
00077             if (!cell_inst->cell_ref) {
00078                 total_unresolved_count++;
00079                 cell->checks.unresolved_child_count++;
00080             }
00081         }
00082     }
00083
00084     return total_unresolved_count;
00085 }
00086
00093 static int gds_tree_check_list_contains_cell(GList *list, struct gds_cell *cell) {
00094     GList *iter;
00095
00096     for (iter = list; iter != NULL; iter = g_list_next(iter)) {
00097         if ((struct gds_cell *)iter->data == cell)
00098             return 1;
00099     }
00100
00101     return 0;
00102 }
00103
00110 static int gds_tree_check_iterate_ref_and_check(struct gds_cell *cell_to_check, GList **visited_cells)
00111 {
00112     GList *ref_iter;
00113     struct gds_cell_instance *ref;

```



```

00114     struct gds_cell *sub_cell;
00115     int res;
00116
00117     if (!cell_to_check)
00118         return -1;
00119
00120     /* Check if this cell is already contained in visited cells. This indicates a loop */
00121     if (gds_tree_check_list_contains_cell(*visited_cells, cell_to_check))
00122         return 1;
00123
00124     /* Add cell to visited cell list */
00125     *visited_cells = g_list_append(*visited_cells, (gpointer)cell_to_check);
00126
00127     /* Mark references and process sub cells */
00128     for (ref_iter = cell_to_check->child_cells; ref_iter != NULL; ref_iter = g_list_next(ref_iter)) {
00129         ref = (struct gds_cell_instance *)ref_iter->data;
00130
00131         if (!ref)
00132             return -1;
00133
00134         sub_cell = ref->cell_ref;
00135
00136         /* If cell is not resolved, ignore. No harm there */
00137         if (!sub_cell)
00138             continue;
00139
00140         res = gds_tree_check_iterate_ref_and_check(sub_cell, visited_cells);
00141         if (res < 0) {
00142             /* Error. return. */
00143             return -3;
00144         } else if (res > 0) {
00145             /* Loop in subcell found. Propagate to top */
00146             return 1;
00147         }
00148     }
00149
00150     /* Remove cell from visited cells */
00151     *visited_cells = g_list_remove(*visited_cells, cell_to_check);
00152
00153     /* No error found in this chain */
00154     return 0;
00155 }
00156
00157 int gds_tree_check_reference_loops(struct gds_library *lib)
00158 {
00159     int res;
00160     int loop_count = 0;
00161     GList *cell_iter;
00162     struct gds_cell *cell_to_check;
00163     GList *visited_cells = NULL;
00164
00165
00166     if (!lib)
00167         return -1;
00168
00169     for (cell_iter = lib->cells; cell_iter != NULL; cell_iter = g_list_next(cell_iter)) {
00170         cell_to_check = (struct gds_cell *)cell_iter->data;
00171
00172         /* A broken cell reference will be counted fatal in this case */
00173         if (!cell_to_check)
00174             return -2;
00175
00176         /* iterate through references and check if loop exists */
00177         res = gds_tree_check_iterate_ref_and_check(cell_to_check, &visited_cells);
00178
00179         if (res < 0) {
00180             /* Error */
00181             return res;
00182         } else if (res > 0) {
00183             /* Loop found: increment loop count and flag cell */
00184             cell_to_check->checks.affected_by_reference_loop = 1;
00185             loop_count++;
00186         } else if (res == 0) {
00187             /* No error found for this cell */
00188             cell_to_check->checks.affected_by_reference_loop = 0;
00189         }
00190     }
00191
00192
00193     if (visited_cells) {
00194         fprintf(stderr, "Visited cell list should be empty. This is a bug. Please report this.\n");
00195         g_list_free(visited_cells);
00196     }
00197
00198     return loop_count;
00199 }
00200

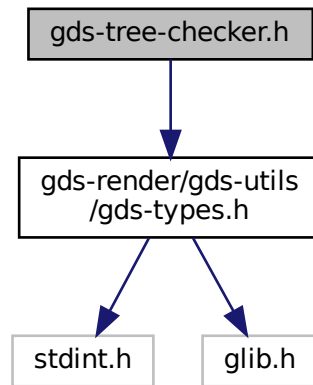
```

13.39 gds-tree-checker.h File Reference

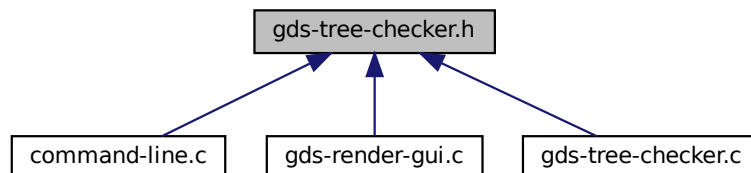
Checking functions of a cell tree (Header)

```
#include <gds-render/gds-utils/gds-types.h>
```

Include dependency graph for gds-tree-checker.h:



This graph shows which files directly or indirectly include this file:



Functions

- int [gds_tree_check_cell_references](#) (struct [gds_library](#) *lib)
gds_tree_check_cell_references checks if all child cell references can be resolved in the given library
- int [gds_tree_check_reference_loops](#) (struct [gds_library](#) *lib)
gds_tree_check_reference_loops checks if the given library contains reference loops

13.39.1 Detailed Description

Checking functions of a cell tree (Header)

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [gds-tree-checker.h](#).

13.40 gds-tree-checker.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef _GDS_TREE_CHECKER_H_
00032 #define _GDS_TREE_CHECKER_H_
00033
00034 #include <gds-render/gds-utils/gds-types.h>
00035
00049 int gds_tree_check_cell_references(struct gds_library *lib);
00050
00057 int gds_tree_check_reference_loops(struct gds_library *lib);
00058
00059 #endif /* _GDS_TREE_CHECKER_H_ */
00060

```

13.41 gds-types.h File Reference

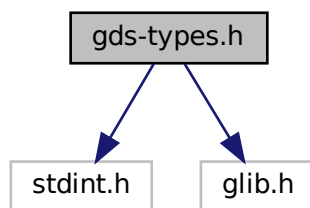
Defines types and macros used by the GDS-Parser.

```

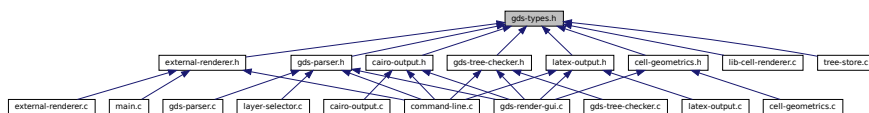
#include <stdint.h>
#include <glib.h>

```

Include dependency graph for gds-types.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [gds_point](#)
A point in the 2D plane. Sometimes references as vertex.
- struct [gds_cell_checks](#)
Stores the result of the cell checks.
- struct [gds_cell_checks::_check_internals](#)
For the internal use of the checker.
- struct [gds_time_field](#)
Date information for cells and libraries.
- struct [gds_graphics](#)
A GDS graphics object.
- struct [gds_cell_instance](#)
This represents an instanc of a cell inside another cell.
- struct [gds_cell](#)
A Cell inside a [gds_library](#).
- struct [gds_library](#)
GDS Toplevel library.

Macros

- #define [CELL_NAME_MAX](#) (100)
Maximum length of a [gds_cell::name](#) or a [gds_library::name](#).
- #define [MIN](#)(a, b) (((a) < (b)) ? (a) : (b))
Return smaller number.
- #define [MAX](#)(a, b) (((a) > (b)) ? (a) : (b))
Return bigger number.

Enumerations

- enum { [GDS_CELL_CHECK_NOT_RUN](#) = -1 }
Definition of check counter default value that indicates that the corresponding check has not yet been executed.
- enum [graphics_type](#) { [GRAPHIC_PATH](#) = 0, [GRAPHIC_POLYGON](#) = 1, [GRAPHIC_BOX](#) = 2 }
Types of graphic objects.
- enum [path_type](#) { [PATH_FLUSH](#) = 0, [PATH_ROUNDED](#) = 1, [PATH_SQUARED](#) = 2 }
Defines the line caps of a path.

13.41.1 Detailed Description

Defines types and macros used by the GDS-Parser.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [gds-types.h](#).

13.42 gds-types.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef __GDS_TYPES_H__
00032 #define __GDS_TYPES_H__
00033
00034 #include <stdint.h>
00035 #include <glib.h>
00036
00037 #define CELL_NAME_MAX (100)
00039 /* Maybe use the macros that ship with the compiler? */
00040 #define MIN(a,b) (((a) < (b)) ? (a) : (b))
00041 #define MAX(a,b) (((a) > (b)) ? (a) : (b))
00045 enum {GDS_CELL_CHECK_NOT_RUN = -1};
00046
00048 enum graphics_type
00049 {
00050     GRAPHIC_PATH = 0,
00051     GRAPHIC_POLYGON = 1,
00052     GRAPHIC_BOX = 2
00053 };
00054
00058 enum path_type {PATH_FLUSH = 0, PATH_ROUNDED = 1, PATH_SQUARED = 2};
00063 struct gds_point {
00064     int x;
00065     int y;
00066 };
00067
00071 struct gds_cell_checks {
00072     int unresolved_child_count;
00073     int affected_by_reference_loop;
00078     struct _check_internals {
00079         int marker;
00080     } _internal;
00081 };
00082
00086 struct gds_time_field {
00087     uint16_t year;
00088     uint16_t month;
00089     uint16_t day;
00090     uint16_t hour;
00091     uint16_t minute;
00092     uint16_t second;
00093 };
00094
00098 struct gds_graphics {
00099     enum graphics_type gfx_type;
00100     GList *vertices;
00101     enum path_type path_render_type;
00102     int width_absolute;
00103     int16_t layer;
00104     uint16_t datatype;
00105 };
00106
00110 struct gds_cell_instance {
00111     char ref_name[CELL_NAME_MAX];
00112     struct gds_cell *cell_ref;
00113     struct gds_point origin;
00114     int flipped;
00115     double angle;
00116     double magnification;
00117 };
00118
00122 struct gds_cell {
00123     char name[CELL_NAME_MAX];
00124     struct gds_time_field mod_time;
00125     struct gds_time_field access_time;
00126     GList *child_cells;
00127     GList *graphic_objs;

```

```

00128     struct gds_library *parent_library;
00129     struct gds_cell_checks checks;
00130 };
00131
00132 struct gds_library {
00133     char name[CELL_NAME_MAX];
00134     struct gds_time_field mod_time;
00135     struct gds_time_field access_time;
00136     double unit_in_meters;
00137     GList *cells;
00138     GList *cell_names ;
00139 };
00140 #endif /* __GDS_TYPES_H__ */

```

13.43 geometric.dox File Reference

13.44 gpl-2.0.md File Reference

13.45 gui.dox File Reference

13.46 latex-output.c File Reference

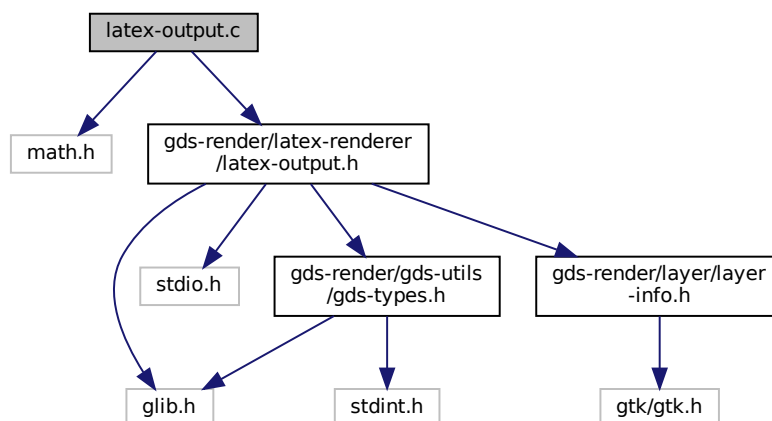
LaTeX output renderer.

```

#include <math.h>
#include <gds-render/latex-renderer/latex-output.h>

```

Include dependency graph for latex-output.c:



Macros

- `#define WRITEOUT_BUFFER(buff) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)`
Writes a *GString* *buffer* to the fixed file *tex_file*.

Functions

- static void `write_layer_definitions` (FILE *tex_file, GList *layer_infos, GString *buffer)
Write the layer declarration to TeX file.
- static gboolean `write_layer_env` (FILE *tex_file, GdkRGBA *color, int layer, GList *linfo, GString *buffer)
Write layer Envirmonment.
- static void `generate_graphics` (FILE *tex_file, GList *graphics, GList *linfo, GString *buffer, double scale)
Writes a graphics object to the specified tex_file.
- static void `render_cell` (struct `gds_cell` *cell, GList *layer_infos, FILE *tex_file, GString *buffer, double scale)
Render cell to file.
- void `latex_render_cell_to_code` (struct `gds_cell` *cell, GList *layer_infos, FILE *tex_file, double scale, gboolean create_pdf_layers, gboolean standalone_document)
Render cell to LateX/TikZ code.

13.46.1 Detailed Description

LaTeX output renderer.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [latex-output.c](#).

13.47 latex-output.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <math.h>
00027
00028 #include <gds-render/latex-renderer/latex-output.h>
00029
00036 #define WRITEOUT_BUFFER(buff) fwrite((buff)->str, sizeof(char), (buff)->len, tex_file)
00037
00050 static void write_layer_definitions(FILE *tex_file, GList *layer_infos, GString *buffer)
00051 {
00052     GList *list;
00053     struct layer_info *lifo;
00054     char *end_str;
00055
00056     for (list = layer_infos; list != NULL; list = list->next) {
00057         lifo = (struct layer_info *)list->data;
00058         g_string_printf(buffer, "\\pgfdeclarelayer{l%d}\n\\definecolor{c%d}{rgb}{%lf,%lf,%lf}\n",
00059             lifo->layer, lifo->layer,
00060             lifo->color.red, lifo->color.green, lifo->color.blue);
00061         WRITEOUT_BUFFER(buffer);
00062     }
00063
00064     g_string_printf(buffer, "\\pgfsetlayers{");

```

```

00065     WRITEOUT_BUFFER(buffer);
00066
00067     for (list = layer_infos; list != NULL; list = list->next) {
00068         lifo = (struct layer_info *)list->data;
00069
00070         if (list->next == NULL)
00071             end_str = ",main>";
00072         else
00073             end_str = ",";
00074         g_string_printf(buffer, "%d", lifo->layer, end_str);
00075         WRITEOUT_BUFFER(buffer);
00076     }
00077     fwrite("\n", sizeof(char), 1, tex_file);
00078 }
00079
00106 static gboolean write_layer_env(FILE *tex_file, GdkgRGBA *color, int layer, GList *linfo, GString
    *buffer)
00107 {
00108     GList *temp;
00109     struct layer_info *inf;
00110
00111     for (temp = linfo; temp != NULL; temp = temp->next) {
00112         inf = (struct layer_info *)temp->data;
00113         if (inf->layer == layer) {
00114             color->alpha = inf->color.alpha;
00115             color->red = inf->color.red;
00116             color->green = inf->color.green;
00117             color->blue = inf->color.blue;
00118             g_string_printf(buffer,
00119 "\begin{pgfonlayer}{%d}\n\\ifcreatepdflayers\n\\begin{scope}[ocg={ref=%d,
status=visible,name={%s}}]\n\\fi\n",
00119             layer, layer, inf->name);
00120             WRITEOUT_BUFFER(buffer);
00121             return TRUE;
00122         }
00123     }
00124     return FALSE;
00125 }
00126
00138 static void generate_graphics(FILE *tex_file, GList *graphics, GList *linfo, GString *buffer, double
    scale)
00139 {
00140     GList *temp;
00141     GList *temp_vertex;
00142     struct gds_graphics *gfx;
00143     struct gds_point *pt;
00144     GdkgRGBA color;
00145     static const char *line_caps[] = {"butt", "round", "rect"};
00146
00147     for (temp = graphics; temp != NULL; temp = temp->next) {
00148         gfx = (struct gds_graphics *)temp->data;
00149         if (write_layer_env(tex_file, &color, (int)gfx->layer, linfo, buffer) == TRUE) {
00150
00151             /* Layer is defined => create graphics */
00152             if (gfx->gfx_type == GRAPHIC_POLYGON || gfx->gfx_type == GRAPHIC_BOX ) {
00153                 g_string_printf(buffer, "\\draw[line width=0.00001 pt, draw={c%d}, fill={c%d}, fill
opacity={%lf}] ",
00154                 gfx->layer, gfx->layer, color.alpha);
00155                 WRITEOUT_BUFFER(buffer);
00156                 /* Append vertices */
00157                 for (temp_vertex = gfx->vertices; temp_vertex != NULL; temp_vertex =
temp_vertex->next) {
00158                     pt = (struct gds_point *)temp_vertex->data;
00159                     g_string_printf(buffer, "%lf pt, %lf pt) -- ", ((double)pt->x)/scale,
((double)pt->y)/scale);
00160                     WRITEOUT_BUFFER(buffer);
00161                 }
00162                 g_string_printf(buffer, "cycle;\n");
00163                 WRITEOUT_BUFFER(buffer);
00164             } else if (gfx->gfx_type == GRAPHIC_PATH) {
00165
00166                 if (g_list_length(gfx->vertices) < 2) {
00167                     printf("Cannot write path with less than 2 points\n");
00168                     break;
00169                 }
00170
00171                 if (gfx->path_render_type < 0 || gfx->path_render_type > 2) {
00172                     printf("Path type unrecognized. Setting to 'flushed'\n");
00173                     gfx->path_render_type = PATH_FLUSH;
00174                 }
00175
00176                 g_string_printf(buffer, "\\draw[line width=%lf pt, draw={c%d}, opacity={%lf}, cap=%s]
",
00177                 gfx->width_absolute/scale, gfx->layer, color.alpha,
00178                 line_caps[gfx->path_render_type]);
00179                 WRITEOUT_BUFFER(buffer);
00180

```



```

00181         /* Append vertices */
00182         for (temp_vertex = gfx->vertices; temp_vertex != NULL; temp_vertex =
temp_vertex->next) {
00183             pt = (struct gds_point *)temp_vertex->data;
00184             g_string_printf(buffer, "%lf pt, %lf pt)%s",
00185                 ((double)pt->x)/scale,
00186                 ((double)pt->y)/scale,
00187                 (temp_vertex->next ? " -- " : ""));
00188             WRITEOUT_BUFFER(buffer);
00189         }
00190         g_string_printf(buffer, ";\n");
00191         WRITEOUT_BUFFER(buffer);
00192     }
00193 }
00194 g_string_printf(buffer, "\\ifcreatepdflayers\n\\end{scope}\n\\fi\n\\end{pgfonlayer}\n");
00195 WRITEOUT_BUFFER(buffer);
00196 }
00197 } /* For graphics */
00198 }
00199 }
00200
00209 static void render_cell(struct gds_cell *cell, GList *layer_infos, FILE *tex_file, GString *buffer,
double scale)
00210 {
00211     GList *list_child;
00212     struct gds_cell_instance *inst;
00213
00214     /* Draw polygons of current cell */
00215     generate_graphics(tex_file, cell->graphic_objs, layer_infos, buffer, scale);
00216
00217     /* Draw polygons of childs */
00218     for (list_child = cell->child_cells; list_child != NULL; list_child = list_child->next) {
00219         inst = (struct gds_cell_instance *)list_child->data;
00220
00221         /* Abort if cell has no reference */
00222         if (!inst->cell_ref)
00223             continue;
00224
00225         /* generate translation scope */
00226         g_string_printf(buffer, "\\begin{scope}[shift={%lf pt,%lf pt}]\n",
00227             ((double)inst->origin.x)/scale, ((double)inst->origin.y)/scale);
00228         WRITEOUT_BUFFER(buffer);
00229
00230         g_string_printf(buffer, "\\begin{scope}[rotate=%lf]\n", inst->angle);
00231         WRITEOUT_BUFFER(buffer);
00232
00233         g_string_printf(buffer, "\\begin{scope}[yscale=%lf, xscale=%lf]\n", (inst->flipped ?
-1*inst->magnification : inst->magnification),
00234             inst->magnification);
00235         WRITEOUT_BUFFER(buffer);
00236
00237         render_cell(inst->cell_ref, layer_infos, tex_file, buffer, scale);
00238
00239         g_string_printf(buffer, "\\end{scope}\n");
00240         WRITEOUT_BUFFER(buffer);
00241
00242         g_string_printf(buffer, "\\end{scope}\n");
00243         WRITEOUT_BUFFER(buffer);
00244
00245         g_string_printf(buffer, "\\end{scope}\n");
00246         WRITEOUT_BUFFER(buffer);
00247     }
00248 }
00249
00250 }
00251
00252 void latex_render_cell_to_code(struct gds_cell *cell, GList *layer_infos, FILE *tex_file, double
scale,
00253     gboolean create_pdf_layers, gboolean standalone_document)
00254 {
00255     GString *working_line;
00256
00257     if (!tex_file || !layer_infos || !cell)
00258         return;
00259
00260     /* 10 kB Line working buffer should be enough */
00261     working_line = g_string_new_len(NULL, LATEX_LINE_BUFFER_KB*1024);
00262
00263     /* standalone foo */
00264     g_string_printf(working_line, "\\newif\\iftestmode\n\\testmode%s\n",
00265         (standalone_document ? "true" : "false"));
00266     WRITEOUT_BUFFER(working_line);
00267     g_string_printf(working_line, "\\newif\\ifcreatepdflayers\n\\createpdflayers%s\n",
00268         (create_pdf_layers ? "true" : "false"));
00269     WRITEOUT_BUFFER(working_line);
00270     g_string_printf(working_line, "\\iftestmode\n");
00271

```

```

00272     WRITEOUT_BUFFER(working_line);
00273     g_string_printf(working_line,
00274     "\\documentclass[tikz]{standalone}\n\\usepackage{xcolor}\n\\usetikzlibrary{ocgx}\n\\begin{document}\n");
00274     WRITEOUT_BUFFER(working_line);
00275     g_string_printf(working_line, "\\fi\n");
00276     WRITEOUT_BUFFER(working_line);
00277
00278     /* Write layer definitions */
00279     write_layer_definitions(tex_file, layer_infos, working_line);
00280
00281     /* Open tikz Pictute */
00282     g_string_printf(working_line, "\\begin{tikzpicture}\n");
00283     WRITEOUT_BUFFER(working_line);
00284
00285     /* Generate graphics output */
00286     render_cell(cell, layer_infos, tex_file, working_line, scale);
00287
00288
00289     g_string_printf(working_line, "\\end{tikzpicture}\n");
00290     WRITEOUT_BUFFER(working_line);
00291
00292     g_string_printf(working_line, "\\iftestmode\n");
00293     WRITEOUT_BUFFER(working_line);
00294     g_string_printf(working_line, "\\end{document}\n");
00295     WRITEOUT_BUFFER(working_line);
00296     g_string_printf(working_line, "\\fi\n");
00297     WRITEOUT_BUFFER(working_line);
00298
00299     fflush(tex_file);
00300     g_string_free(working_line, TRUE);
00301 }
00302

```

13.48 latex-output.h File Reference

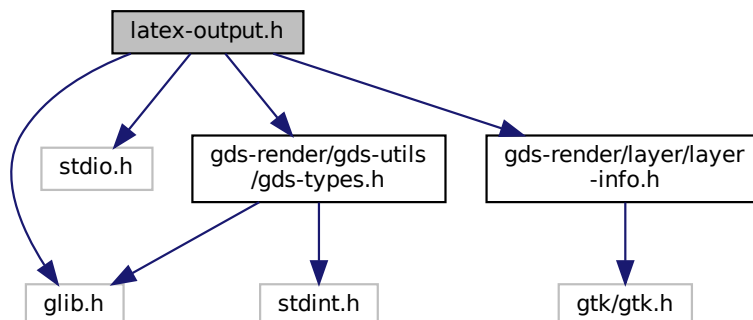
LaTeX output renderer.

```

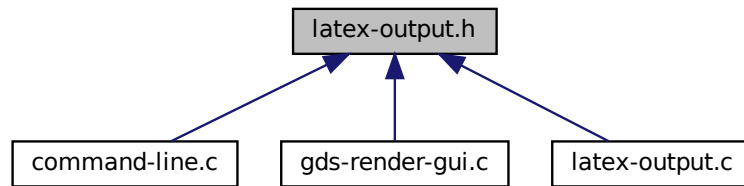
#include <glib.h>
#include <stdio.h>
#include "gds-render/layer/layer-info.h"
#include <gds-render/gds-utils/gds-types.h>

```

Include dependency graph for latex-output.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define LATEX_LINE_BUFFER_KB (10)`
Buffer for LaTeX Code line in KiB.

Functions

- void `latex_render_cell_to_code` (struct `gds_cell` *cell, GList *layer_infos, FILE *tex_file, double scale, gboolean create_pdf_layers, gboolean standalone_document)
Render cell to LaTeX/TikZ code.

13.48.1 Detailed Description

LaTeX output renderer.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [latex-output.h](#).

13.49 latex-output.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
  
```

```

00019
00026 #ifndef __LATEX_OUTPUT_H__
00027 #define __LATEX_OUTPUT_H__
00028
00034 #include <glib.h>
00035 #include <stdio.h>
00036
00037 #include "gds-render/layer/layer-info.h"
00038 #include <gds-render/gds-utils/gds-types.h>
00039
00040 #define LATEX_LINE_BUFFER_KB (10)
00051 void latex_render_cell_to_code(struct gds_cell *cell, GList *layer_infos, FILE *tex_file, double
    scale,
00052                               gboolean create_pdf_layers, gboolean standalone_document);
00053
00056 #endif /* __LATEX_OUTPUT_H__ */

```

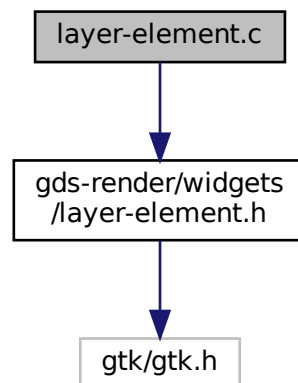
13.50 latex-renderer.dox File Reference

13.51 layer-element.c File Reference

Implementation of the layer element used for configuring layer colors etc.

```
#include <gds-render/widgets/layer-element.h>
```

Include dependency graph for layer-element.c:



Functions

- static void [layer_element_dispose](#) (GObject *obj)
- static void [layer_element_constructed](#) (GObject *obj)
- static void [layer_element_class_init](#) (LayerElementClass *klass)
- static void [layer_element_init](#) (LayerElement *self)
- GtkWidget * [layer_element_new](#) (void)
 - Create new layer element object.*
- const char * [layer_element_get_name](#) (LayerElement *elem)
 - get name of the layer*

- void `layer_element_set_name` (LayerElement *elem, const char *name)
layer_element_set_name
- void `layer_element_set_layer` (LayerElement *elem, int layer)
Set layer number for this layer.
- int `layer_element_get_layer` (LayerElement *elem)
Get layer number.
- void `layer_element_set_export` (LayerElement *elem, gboolean export)
Set export flag for this layer.
- gboolean `layer_element_get_export` (LayerElement *elem)
Get export flag of layer.
- void `layer_element_get_color` (LayerElement *elem, GdkRGBA *rgba)
Get color of layer.
- void `layer_element_set_color` (LayerElement *elem, GdkRGBA *rgba)
Set color of layer.
- void `layer_element_set_dnd_callbacks` (LayerElement *elem, struct `layer_element_dnd_data` *data)
Setup drag and drop of elem for use in the LayerSelector.

13.51.1 Detailed Description

Implementation of the layer element used for configuring layer colors etc.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [layer-element.c](#).

13.52 layer-element.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020 /*
00021  * The drag and drop implementation is adapted from
00022  * https://gitlab.gnome.org/GNOME/gtk/blob/gtk-3-22/tests/testlist3.c
00023  *
00024  * Thanks to the GTK3 people for creating these examples.
00025  */
00026
00039 #include <gds-render/widgets/layer-element.h>
00040
00041 G_DEFINE_TYPE(LayerElement, layer_element, GTK_TYPE_LIST_BOX_ROW)
00042
00043 static void layer_element_dispose(GObject *obj)
00044 {
00045     /* destroy parent container. This destroys all widgets inside */
00046     G_OBJECT_CLASS(layer_element_parent_class)->dispose(obj);

```

```

00047 }
00048
00049 static void layer_element_constructed(GObject *obj)
00050 {
00051     G_OBJECT_CLASS(layer_element_parent_class)->constructed(obj);
00052 }
00053
00054 static void layer_element_class_init(LayerElementClass *klass)
00055 {
00056     GObjectClass *oclass = G_OBJECT_CLASS(klass);
00057     oclass->dispose = layer_element_dispose;
00058     oclass->constructed = layer_element_constructed;
00059 }
00060
00061 static void layer_element_init(LayerElement *self)
00062 {
00063     GtkBuilder *builder;
00064     GtkWidget *glade_box;
00065     builder = gtk_builder_new_from_resource("/layer-widget.glade");
00066     glade_box = GTK_WIDGET(gtk_builder_get_object(builder, "box"));
00067     gtk_container_add(GTK_CONTAINER(self), glade_box);
00068
00069     /* Get Elements */
00070     self->priv.color = GTK_COLOR_BUTTON(gtk_builder_get_object(builder, "color"));
00071     self->priv.export = GTK_CHECK_BUTTON(gtk_builder_get_object(builder, "export"));
00072     self->priv.layer = GTK_LABEL(gtk_builder_get_object(builder, "layer"));
00073     self->priv.name = GTK_ENTRY(gtk_builder_get_object(builder, "entry"));
00074     self->priv.event_handle = GTK_EVENT_BOX(gtk_builder_get_object(builder, "event-box"));
00075
00076     g_object_unref(builder);
00077 }
00078
00079 GtkWidget *layer_element_new(void)
00080 {
00081     return GTK_WIDGET(g_object_new(TYPE_LAYER_ELEMENT, NULL));
00082 }
00083
00084 const char *layer_element_get_name(LayerElement *elem)
00085 {
00086     return gtk_entry_get_text(elem->priv.name);
00087 }
00088
00089 void layer_element_set_name(LayerElement *elem, const char* name)
00090 {
00091     gtk_entry_set_text(elem->priv.name, name);
00092 }
00093
00094 void layer_element_set_layer(LayerElement *elem, int layer)
00095 {
00096     GString *string;
00097
00098     string = g_string_new_len(NULL, 100);
00099     g_string_printf(string, "Layer: %d", layer);
00100     gtk_label_set_text(elem->priv.layer, (const gchar *)string->str);
00101     elem->priv.layer_num = layer;
00102     g_string_free(string, TRUE);
00103 }
00104
00105 int layer_element_get_layer(LayerElement *elem)
00106 {
00107     return elem->priv.layer_num;
00108 }
00109
00110 void layer_element_set_export(LayerElement *elem, gboolean export)
00111 {
00112     gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(elem->priv.export), export);
00113 }
00114
00115 gboolean layer_element_get_export(LayerElement *elem)
00116 {
00117     return gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(elem->priv.export));
00118 }
00119
00120 void layer_element_get_color(LayerElement *elem, GdkRGBA *rgba)
00121 {
00122     if (!rgba)
00123         return;
00124
00125     gtk_color_chooser_get_rgba(GTK_COLOR_CHOOSER(elem->priv.color), rgba);
00126 }
00127
00128 void layer_element_set_color(LayerElement *elem, GdkRGBA *rgba)
00129 {
00130     if (!elem || !rgba)
00131         return;
00132
00133     gtk_color_chooser_set_rgba(GTK_COLOR_CHOOSER(elem->priv.color), rgba);

```

```

00134 }
00135
00136 void layer_element_set_dnd_callbacks(LayerElement *elem, struct layer_element_dnd_data *data)
00137 {
00138     if (!elem || !data)
00139         return;
00140
00141     /* Setup drag and drop */
00142     gtk_style_context_add_class (gtk_widget_get_style_context(GTK_WIDGET(elem)), "row");
00143     gtk_drag_source_set(GTK_WIDGET(elem->priv.event_handle), GDK_BUTTON1_MASK, data->entries,
data->entry_count, GDK_ACTION_MOVE);
00144     g_signal_connect(elem->priv.event_handle, "drag-begin", G_CALLBACK(data->drag_begin), NULL);
00145     g_signal_connect(elem->priv.event_handle, "drag-data-get", G_CALLBACK(data->drag_data_get), NULL);
00146     g_signal_connect(elem->priv.event_handle, "drag-end", G_CALLBACK(data->drag_end), NULL);
00147
00148 }
00149

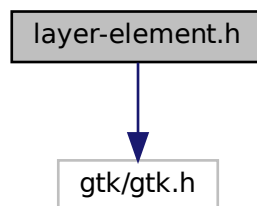
```

13.53 layer-element.h File Reference

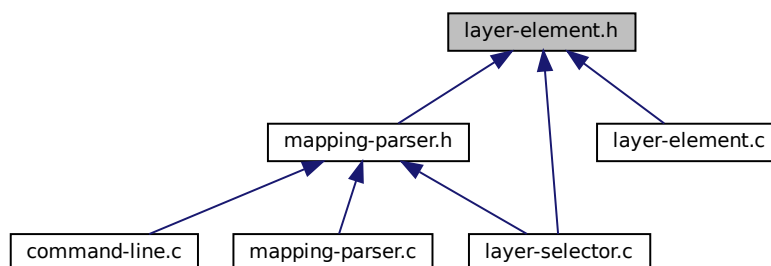
Implementation of the layer element used for configuring layer colors etc.

```
#include <gtk/gtk.h>
```

Include dependency graph for layer-element.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [_LayerElementPriv](#)
- struct [_LayerElement](#)
- struct [layer_element_dnd_data](#)

This structure holds the necessary data to set up a LayerElement for Drag'n'Drop.

Macros

- #define [TYPE_LAYER_ELEMENT](#) ([layer_element_get_type\(\)](#))

Typedefs

- typedef struct [_LayerElementPriv](#) [LayerElementPriv](#)

Functions

- GtkWidget * [layer_element_new](#) (void)
Create new layer element object.
- const char * [layer_element_get_name](#) (LayerElement *elem)
get name of the layer
- void [layer_element_set_name](#) (LayerElement *elem, const char *name)
layer_element_set_name
- void [layer_element_set_layer](#) (LayerElement *elem, int layer)
Set layer number for this layer.
- int [layer_element_get_layer](#) (LayerElement *elem)
Get layer number.
- void [layer_element_set_export](#) (LayerElement *elem, gboolean export)
Set export flag for this layer.
- gboolean [layer_element_get_export](#) (LayerElement *elem)
Get export flag of layer.
- void [layer_element_get_color](#) (LayerElement *elem, GdkRGBA *rgba)
Get color of layer.
- void [layer_element_set_color](#) (LayerElement *elem, GdkRGBA *rgba)
Set color of layer.
- void [layer_element_set_dnd_callbacks](#) (LayerElement *elem, struct [layer_element_dnd_data](#) *data)
Setup drag and drop of elem for use in the LayerSelector.

13.53.1 Detailed Description

Implementation of the layer element used for configuring layer colors etc.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [layer-element.h](#).

13.54 layer-element.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef __LAYER_ELEMENT_H__
00033 #define __LAYER_ELEMENT_H__
00034
00035 #include <gtk/gtk.h>
00036
00037 G_BEGIN_DECLS
00038
00039 /* Creates Class structure etc */
00040 G_DECLARE_FINAL_TYPE(LayerElement, layer_element, LAYER, ELEMENT, GtkListBoxRow)
00041
00042 #define TYPE_LAYER_ELEMENT (layer_element_get_type())
00043
00044 typedef struct _LayerElementPriv {
00045     GtkEntry *name;
00046     GtkLabel *layer;
00047     int layer_num;
00048     GtkEventBox *event_handle;
00049     GtkColorButton *color;
00050     GtkCheckButton *export;
00051 } LayerElementPriv;
00052
00053 struct _LayerElement {
00054     /* Inheritance */
00055     GtkListBoxRow parent;
00056     /* Custom Elements */
00057     LayerElementPriv priv;
00058 };
00059
00063 struct layer_element_dnd_data {
00065     GtkTargetEntry *entries;
00067     int entry_count;
00069     void (*drag_begin)(GtkWidget *, GdkDragContext *, gpointer);
00071     void (*drag_data_get)(GtkWidget *, GdkDragContext *, GtkSelectionData *, guint, guint, gpointer);
00073     void (*drag_end)(GtkWidget *, GdkDragContext *, gpointer);
00074 };
00075
00080 GtkWidget *layer_element_new(void);
00081
00087 const char *layer_element_get_name(LayerElement *elem);
00088
00094 void layer_element_set_name(LayerElement *elem, const char * name);
00095
00101 void layer_element_set_layer(LayerElement *elem, int layer);
00102
00108 int layer_element_get_layer(LayerElement *elem);
00109
00115 void layer_element_set_export(LayerElement *elem, gboolean export);
00116
00122 gboolean layer_element_get_export(LayerElement *elem);
00123
00129 void layer_element_get_color(LayerElement *elem, GdkRGBA *rgba);
00130
00136 void layer_element_set_color(LayerElement *elem, GdkRGBA *rgba);
00137
00143 void layer_element_set_dnd_callbacks(LayerElement *elem, struct layer_element_dnd_data *data);
00144
00145 G_END_DECLS
00146
00147 #endif /* __LAYER_ELEMENT_H__ */
00148

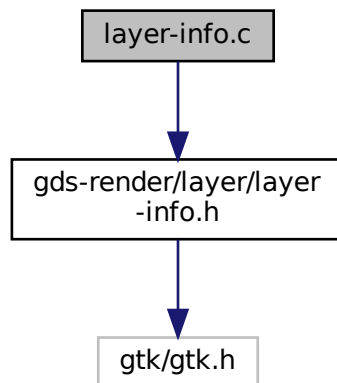
```

13.55 layer-info.c File Reference

Helper functions for layer info struct.

```
#include <gds-render/layer/layer-info.h>
```

Include dependency graph for layer-info.c:



Functions

- void [layer_info_delete_struct](#) (struct [layer_info](#) *info)
Delete a [layer_info](#) struct.

13.55.1 Detailed Description

Helper functions for layer info struct.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [layer-info.c](#).

13.55.2 Function Documentation

13.55.2.1 layer_info_delete_struct()

```
void layer_info_delete_struct (  
    struct layer\_info * info )
```

Delete a [layer_info](#) struct.

Parameters

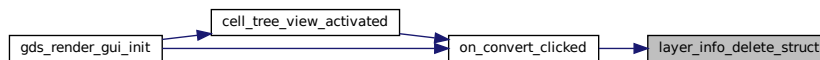
<i>info</i>	Struct to be deleted.
-------------	-----------------------

Note

The `layer_info::name` Element has to be freed manually

Definition at line 28 of file `layer-info.c`.

Here is the caller graph for this function:



13.56 layer-info.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <gds-render/layer/layer-info.h>
00027
00028 void layer_info_delete_struct(struct layer_info *info)
00029 {
00030     if (info)
00031         free(info);
00032 }
00033

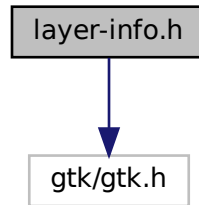
```

13.57 layer-info.h File Reference

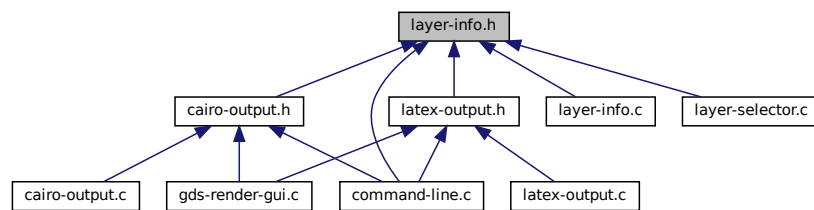
Helper functions and definition of layer info struct.

```
#include <gtk/gtk.h>
```

Include dependency graph for layer-info.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [layer_info](#)
Layer information.

Functions

- void [layer_info_delete_struct](#) (struct [layer_info](#) *info)
Delete a [layer_info](#) struct.

13.57.1 Detailed Description

Helper functions and definition of layer info struct.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [layer-info.h](#).

13.57.2 Function Documentation

13.57.2.1 layer_info_delete_struct()

```
void layer_info_delete_struct (
    struct layer_info * info )
```

Delete a `layer_info` struct.

Parameters

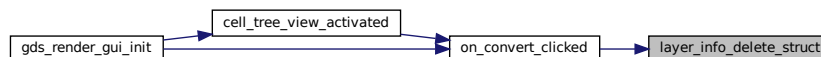
<code>info</code>	Struct to be deleted.
-------------------	-----------------------

Note

The `layer_info::name` Element has to be freed manually

Definition at line 28 of file `layer-info.c`.

Here is the caller graph for this function:



13.58 layer-info.h

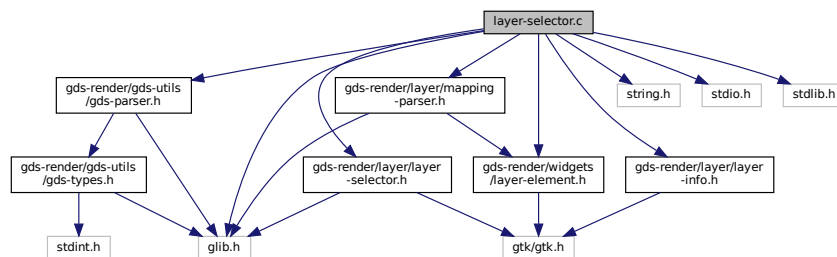
```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2019 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef _LAYER_INFO_H_
00027 #define _LAYER_INFO_H_
00028
00029 #include <gtk/gtk.h>
00030
00036 struct layer_info
00037 {
00038     int layer;
00039     char *name;
00040     int stacked_position;
00041     GdkRGBA color;
00042 };
00043
00049 void layer_info_delete_struct(struct layer_info *info);
00050
00051 #endif // _LAYER_INFO_H_
```

13.59 layer-selector.c File Reference

Implementation of the layer selector.

```
#include <glib.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <gds-render/layer/layer-selector.h>
#include <gds-render/layer/layer-info.h>
#include <gds-render/gds-utils/gds-parser.h>
#include <gds-render/widgets/layer-element.h>
#include <gds-render/layer/mapping-parser.h>
```

Include dependency graph for layer-selector.c:



Data Structures

- struct [_LayerSelector](#)

Functions

- static void [sel_layer_element_drag_begin](#) (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void [sel_layer_element_drag_end](#) (GtkWidget *widget, GdkDragContext *context, gpointer data)
- static void [sel_layer_element_drag_data_get](#) (GtkWidget *widget, GdkDragContext *context, GtkSelectionData *selection_data, guint info, guint time, gpointer data)
- static GtkWidget * [layer_selector_get_last_row](#) (GtkListBox *list)
- static GtkWidget * [layer_selector_get_row_before](#) (GtkListBox *list, GtkWidget *row)
- static GtkWidget * [layer_selector_get_row_after](#) (GtkListBox *list, GtkWidget *row)
- static void [layer_selector_drag_data_received](#) (GtkWidget *widget, GdkDragContext *context, gint x, gint y, GtkSelectionData *selection_data, guint info, guint32 time, gpointer data)
- static gboolean [layer_selector_drag_motion](#) (GtkWidget *widget, GdkDragContext *context, int x, int y, guint time)
- static void [layer_selector_drag_leave](#) (GtkWidget *widget, GdkDragContext *context, guint time)
- static void [layer_selector_dispose](#) (GObject *self)
- static void [layer_selector_class_init](#) (LayerSelectorClass *klass)
- static void [layer_selector_setup_dnd](#) (LayerSelector *self)
- static void [layer_selector_init](#) (LayerSelector *self)
- LayerSelector * [layer_selector_new](#) (GtkListBox *list_box)
 - layer_selector_new*
- GList * [layer_selector_export_rendered_layer_info](#) (LayerSelector *selector)

- Get a list of all layers that shall be exported when rendering the cells.*
- static void [layer_selector_clear_widgets](#) (LayerSelector *self)
 - static gboolean [layer_selector_check_if_layer_widget_exists](#) (LayerSelector *self, int layer)
Check if a specific layer element with the given layer number is present in the layer selector.
 - static void [sel_layer_element_setup_dnd_callbacks](#) (LayerSelector *self, LayerElement *element)
Setup the necessary drag and drop callbacks of layer elements.
 - static void [layer_selector_analyze_cell_layers](#) (LayerSelector *self, struct [gds_cell](#) *cell)
Analyze `cell` layers and append detected layers to layer selector `self`.
 - static gint [layer_selector_sort_func](#) (GtkListBoxRow *row1, GtkListBoxRow *row2, gpointer unused)
sort_func Sort callback for list box
 - void [layer_selector_generate_layer_widgets](#) (LayerSelector *selector, GList *libs)
Generate layer widgets in in the LayerSelector instance.
 - static LayerElement * [layer_selector_find_layer_element_in_list](#) (GList *el_list, int layer)
Find LayerElement in list with specified layer number.
 - static void [layer_selector_load_layer_mapping_from_file](#) (LayerSelector *self, gchar *file_name)
Load the layer mapping from a CSV formatted file.
 - static void [layer_selector_load_mapping_clicked](#) (GtkWidget *button, gpointer user_data)
Callback for Load Mapping Button.
 - static void [layer_selector_save_layer_mapping_data](#) (LayerSelector *self, const gchar *file_name)
Save layer mapping of selector `self` to a file.
 - static void [layer_selector_save_mapping_clicked](#) (GtkWidget *button, gpointer user_data)
Callback for Save Layer Mapping Button.
 - void [layer_selector_set_load_mapping_button](#) (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)
Supply button for loading the layer mapping.
 - void [layer_selector_set_save_mapping_button](#) (LayerSelector *selector, GtkWidget *button, GtkWindow *main_window)
Supply button for saving the layer mapping.
 - void [layer_selector_force_sort](#) (LayerSelector *selector, enum [layer_selector_sort_algo](#) sort_function)
Force the layer selector list to be sorted according to `sort_function`.

Variables

- static const char * [dnd_additional_css](#)

13.59.1 Detailed Description

Implementation of the layer selector.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [layer-selector.c](#).

13.60 layer-selector.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <glib.h>
00032 #include <string.h>
00033 #include <stdio.h>
00034 #include <stdlib.h>
00035
00036 #include <gds-render/layer/layer-selector.h>
00037 #include <gds-render/layer/layer-info.h>
00038 #include <gds-render/gds-utils/gds-parser.h>
00039 #include <gds-render/widgets/layer-element.h>
00040 #include <gds-render/layer/mapping-parser.h>
00041
00042 struct _LayerSelector {
00043     /* Parent */
00044     GObject parent;
00045     /* Own fields */
00046     GtkWidget *associated_load_button;
00047     GtkWidget *associated_save_button;
00048     GtkWindow *load_parent_window;
00049     GtkWindow *save_parent_window;
00050     GtkListBox *list_box;
00051
00052     GtkTargetEntry dnd_target;
00053
00054     gpointer dummy[4];
00055 };
00056
00057 G_DEFINE_TYPE(LayerSelector, layer_selector, G_TYPE_OBJECT)
00058
00059 /* Drag and drop code
00060  * Original code from https://blog.gtk.org/2017/06/01/drag-and-drop-in-lists-revisited/
00061  */
00062
00063 static void sel_layer_element_drag_begin(GtkWidget *widget, GdkDragContext *context, gpointer data)
00064 {
00065     GtkWidget *row;
00066     GtkAllocation alloc;
00067     cairo_surface_t *surface;
00068     cairo_t *cr;
00069     int x, y;
00070     (void)data;
00071
00072     row = gtk_widget_get_ancestor(widget, GTK_TYPE_LIST_BOX_ROW);
00073     gtk_widget_get_allocation(row, &alloc);
00074     surface = cairo_image_surface_create(CAIRO_FORMAT_ARGB32, alloc.width, alloc.height);
00075     cr = cairo_create(surface);
00076
00077     gtk_style_context_add_class(gtk_widget_get_style_context(row), "drag-icon");
00078     gtk_widget_draw(row, cr);
00079     gtk_style_context_remove_class(gtk_widget_get_style_context(row), "drag-icon");
00080
00081     gtk_widget_translate_coordinates(widget, row, 0, 0, &x, &y);
00082     cairo_surface_set_device_offset(surface, -x, -y);
00083     gtk_drag_set_icon_surface(context, surface);
00084
00085     cairo_destroy(cr);
00086     cairo_surface_destroy(surface);
00087
00088     g_object_set_data(G_OBJECT(gtk_widget_get_parent(row)), "drag-row", row);
00089     gtk_style_context_add_class(gtk_widget_get_style_context(row), "drag-row");
00090 }
00091
00092 static void sel_layer_element_drag_end(GtkWidget *widget, GdkDragContext *context, gpointer data)
00093 {
00094     GtkWidget *row;
00095     (void)context;
00096     (void)data;

```



```

00097
00098     row = gtk_widget_get_ancestor(widget, GTK_TYPE_LIST_BOX_ROW);
00099     g_object_set_data(G_OBJECT(gtk_widget_get_parent(row)), "drag-row", NULL);
00100     gtk_style_context_remove_class(gtk_widget_get_style_context(row), "drag-row");
00101     gtk_style_context_remove_class(gtk_widget_get_style_context(row), "drag-hover");
00102 }
00103
00104 static void sel_layer_element_drag_data_get(GtkWidget *widget, GdkDragContext *context,
00105                                             GtkSelectionData *selection_data,
00106                                             guint info, guint time, gpointer data)
00107 {
00108     (void)context;
00109     (void)info;
00110     (void)time;
00111     (void)data;
00112     GdkAtom atom;
00113
00114     atom = gdk_atom_intern_static_string("GTK_LIST_BOX_ROW");
00115
00116     gtk_selection_data_set(selection_data, atom,
00117                           32, (const gchar *)&widget, sizeof(gpointer));
00118 }
00119
00120 static GtkWidget *layer_selector_get_last_row (GtkListBox *list)
00121 {
00122     int i;
00123     GtkWidget *row;
00124
00125     row = NULL;
00126     for (i = 0; ; i++) {
00127         GtkWidget *tmp;
00128         tmp = gtk_list_box_get_row_at_index(list, i);
00129         if (tmp == NULL)
00130             break;
00131         row = tmp;
00132     }
00133
00134     return row;
00135 }
00136
00137 static GtkWidget *layer_selector_get_row_before (GtkListBox *list, GtkWidget *row)
00138 {
00139     int pos;
00140
00141     pos = gtk_list_box_row_get_index (row);
00142     return gtk_list_box_get_row_at_index (list, pos - 1);
00143 }
00144
00145 static GtkWidget *layer_selector_get_row_after (GtkListBox *list, GtkWidget *row)
00146 {
00147     int pos;
00148
00149     pos = gtk_list_box_row_get_index(row);
00150     return gtk_list_box_get_row_at_index(list, pos + 1);
00151 }
00152
00153 static void layer_selector_drag_data_received(GtkWidget *widget, GdkDragContext *context, gint x, gint
00154 y,
00155                                             GtkSelectionData *selection_data, guint info, guint32 time,
00156                                             gpointer data)
00157 {
00158     GtkWidget *row_before, *row_after;
00159     GtkWidget *row;
00160     GtkWidget *source;
00161     int pos;
00162
00163     row_before = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-before"));
00164     row_after = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-after"));
00165
00166     g_object_set_data(G_OBJECT(widget), "row-before", NULL);
00167     g_object_set_data(G_OBJECT(widget), "row-after", NULL);
00168
00169     if (row_before)
00170         gtk_style_context_remove_class(gtk_widget_get_style_context(row_before), "drag-hover-bottom");
00171     if (row_after)
00172         gtk_style_context_remove_class(gtk_widget_get_style_context(row_after), "drag-hover-top");
00173
00174     row = (gpointer) *((gpointer *)gtk_selection_data_get_data(selection_data));
00175     source = gtk_widget_get_ancestor(row, GTK_TYPE_LIST_BOX_ROW);
00176
00177     if (source == row_after)
00178         return;
00179
00180     g_object_ref(source);
00181     gtk_container_remove(GTK_CONTAINER(gtk_widget_get_parent(source)), source);
00182
00183     if (row_after)

```

```

00183     pos = gtk_list_box_row_get_index(GTK_LIST_BOX_ROW(row_after));
00184     else
00185     pos = gtk_list_box_row_get_index(GTK_LIST_BOX_ROW(row_before)) + 1;
00186
00187     gtk_list_box_insert(GTK_LIST_BOX(widget), source, pos);
00188     g_object_unref(source);
00189 }
00190
00191 static gboolean layer_selector_drag_motion(GtkWidget *widget, GdkDragContext *context, int x, int y,
    quint time)
00192 {
00193     GtkAllocation alloc;
00194     GtkWidget *row;
00195     int hover_row_y;
00196     int hover_row_height;
00197     GtkWidget *drag_row;
00198     GtkWidget *row_before;
00199     GtkWidget *row_after;
00200
00201     row = GTK_WIDGET(gtk_list_box_get_row_at_y(GTK_LIST_BOX(widget), y));
00202
00203     drag_row = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "drag-row"));
00204     row_after = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-after"));
00205     row_before = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-before"));
00206
00207     gtk_style_context_remove_class(gtk_widget_get_style_context(drag_row), "drag-hover");
00208     if (row_before)
00209         gtk_style_context_remove_class(gtk_widget_get_style_context(row_before), "drag-hover-bottom");
00210     if (row_after)
00211         gtk_style_context_remove_class(gtk_widget_get_style_context(row_after), "drag-hover-top");
00212
00213     if (row) {
00214         gtk_widget_get_allocation(row, &alloc);
00215         hover_row_y = alloc.y;
00216         hover_row_height = alloc.height;
00217
00218         if (y < hover_row_y + hover_row_height/2) {
00219             row_after = row;
00220             row_before = GTK_WIDGET(layer_selector_get_row_before(GTK_LIST_BOX(widget),
00221                 GTK_LIST_BOX_ROW(row)));
00222         } else {
00223             row_before = row;
00224             row_after = GTK_WIDGET(layer_selector_get_row_after(GTK_LIST_BOX(widget),
00225                 GTK_LIST_BOX_ROW(row)));
00226         }
00227     } else {
00228         row_before = GTK_WIDGET(layer_selector_get_last_row(GTK_LIST_BOX(widget)));
00229         row_after = NULL;
00230     }
00231
00232     g_object_set_data(G_OBJECT(widget), "row-before", row_before);
00233     g_object_set_data(G_OBJECT(widget), "row-after", row_after);
00234
00235     if (drag_row == row_before || drag_row == row_after) {
00236         gtk_style_context_add_class(gtk_widget_get_style_context(drag_row), "drag-hover");
00237         return FALSE;
00238     }
00239
00240     if (row_before)
00241         gtk_style_context_add_class(gtk_widget_get_style_context(row_before), "drag-hover-bottom");
00242     if (row_after)
00243         gtk_style_context_add_class(gtk_widget_get_style_context(row_after), "drag-hover-top");
00244
00245     return TRUE;
00246 }
00247
00248 static void layer_selector_drag_leave(GtkWidget *widget, GdkDragContext *context, quint time)
00249 {
00250     GtkWidget *drag_row;
00251     GtkWidget *row_before;
00252     GtkWidget *row_after;
00253
00254     drag_row = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "drag-row"));
00255     row_before = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-before"));
00256     row_after = GTK_WIDGET(g_object_get_data(G_OBJECT(widget), "row-after"));
00257
00258     gtk_style_context_remove_class(gtk_widget_get_style_context(drag_row), "drag-hover");
00259     if (row_before)
00260         gtk_style_context_remove_class(gtk_widget_get_style_context(row_before), "drag-hover-bottom");
00261     if (row_after)
00262         gtk_style_context_remove_class(gtk_widget_get_style_context(row_after), "drag-hover-top");
00263
00264 }
00265
00266 static const char *dnd_additional_css =
00267     ".row:not(:first-child) { "
00268     "    border-top: 1px solid alpha(gray,0.5); "

```

```

00269 " border-bottom: 1px solid transparent; "
00270 "}"
00271 ".row:first-child { "
00272 " border-top: 1px solid transparent; "
00273 " border-bottom: 1px solid transparent; "
00274 "}"
00275 ".row:last-child { "
00276 " border-top: 1px solid alpha(gray,0.5); "
00277 " border-bottom: 1px solid alpha(gray,0.5); "
00278 "}"
00279 ".row.drag-icon { "
00280 " background: #282828; "
00281 " border: 1px solid blue; "
00282 "}"
00283 ".row.drag-row { "
00284 " color: gray; "
00285 " background: alpha(gray,0.2); "
00286 "}"
00287 ".row.drag-row.drag-hover { "
00288 " border-top: 1px solid #4e9a06; "
00289 " border-bottom: 1px solid #4e9a06; "
00290 "}"
00291 ".row.drag-hover image, "
00292 ".row.drag-hover label { "
00293 " color: #4e9a06; "
00294 "}"
00295 ".row.drag-hover-top {"
00296 " border-top: 1px solid #4e9a06; "
00297 "}"
00298 ".row.drag-hover-bottom {"
00299 " border-bottom: 1px solid #4e9a06; "
00300 "}"
00301
00302 static void layer_selector_dispose(GObject *self)
00303 {
00304     LayerSelector *sel = LAYER_SELECTOR(self);
00305
00306     g_clear_object(&sel->list_box);
00307     g_clear_object(&sel->load_parent_window);
00308     g_clear_object(&sel->save_parent_window);
00309     g_clear_object(&sel->associated_load_button);
00310     g_clear_object(&sel->associated_save_button);
00311
00312     if (sel->dnd_target.target) {
00313         g_free(sel->dnd_target.target);
00314         sel->dnd_target.target = NULL;
00315     }
00316
00317     /* Chain up to parent's dispose function */
00318     G_OBJECT_CLASS(layer_selector_parent_class)->dispose(self);
00319 }
00320
00321 static void layer_selector_class_init(LayerSelectorClass *klass)
00322 {
00323     GObjectClass *object_class = G_OBJECT_CLASS(klass);
00324     GtkCssProvider *provider;
00325
00326     /* Implement handles to virtual functions */
00327     object_class->dispose = layer_selector_dispose;
00328
00329     /* Setup the CSS provider for the drag and drop animations once */
00330     provider = gtk_css_provider_new();
00331     gtk_css_provider_load_from_data(provider, dnd_additional_css, -1, NULL);
00332     gtk_style_context_add_provider_for_screen(gdk_screen_get_default(), GTK_STYLE_PROVIDER(provider),
00333     800);
00334     g_object_unref(provider);
00335 }
00336
00337 static void layer_selector_setup_dnd(LayerSelector *self)
00338 {
00339     gtk_drag_dest_set(GTK_WIDGET(self->list_box), GTK_DEST_DEFAULT_MOTION | GTK_DEST_DEFAULT_DROP,
00340     &self->dnd_target, 1, GDK_ACTION_MOVE);
00341     g_signal_connect(self->list_box, "drag-data-received",
00342     G_CALLBACK(layer_selector_drag_data_received), NULL);
00343     g_signal_connect(self->list_box, "drag-motion", G_CALLBACK(layer_selector_drag_motion), NULL);
00344     g_signal_connect(self->list_box, "drag-leave", G_CALLBACK(layer_selector_drag_leave), NULL);
00345 }
00346 /* Drag and drop end */
00347
00348 static void layer_selector_init(LayerSelector *self)
00349 {
00350     self->load_parent_window = NULL;
00351     self->save_parent_window = NULL;
00352     self->associated_load_button = NULL;
00353     self->associated_save_button = NULL;

```

```

00353
00354     self->dnd_target.target = g_strdup_printf("LAYER_SELECTOR_DND_%p", self);
00355     self->dnd_target.info = 0;
00356     self->dnd_target.flags = GTK_TARGET_SAME_APP;
00357 }
00358
00359 LayerSelector *layer_selector_new(GtkListBox *list_box)
00360 {
00361     LayerSelector *selector;
00362
00363     if (GTK_IS_LIST_BOX(list_box) == FALSE)
00364         return NULL;
00365
00366     selector = LAYER_SELECTOR(g_object_new(TYPE_LAYER_SELECTOR, NULL));
00367     selector->list_box = list_box;
00368     layer_selector_setup_dnd(selector);
00369     g_object_ref(G_OBJECT(list_box));
00370
00371     return selector;
00372 }
00373
00374 GList *layer_selector_export_rendered_layer_info(LayerSelector *selector)
00375 {
00376     GList *info_list = NULL;
00377     LayerElement *le;
00378     struct layer_info *linfo;
00379     GList *row_list;
00380     GList *temp;
00381     int i;
00382
00383     if (!selector)
00384         return NULL;
00385
00386     row_list = gtk_container_get_children(GTK_CONTAINER(selector->list_box));
00387
00388     /* Iterate through widgets and add layers that shall be exported */
00389     for (i = 0, temp = row_list; temp != NULL; temp = temp->next, i++) {
00390
00391         le = LAYER_ELEMENT(temp->data);
00392
00393         if (layer_element_get_export(le) == TRUE) {
00394             /* Allocate new info and fill with info */
00395             linfo = (struct layer_info *)malloc(sizeof(struct layer_info));
00396             layer_element_get_color(le, &linfo->color);
00397             linfo->layer = layer_element_get_layer(le);
00398             linfo->stacked_position = i;
00399             linfo->name = (char *)layer_element_get_name(le);
00400
00401             /* Append to list */
00402             info_list = g_list_append(info_list, (gpointer)linfo);
00403         }
00404     }
00405
00406     return info_list;
00407 }
00408
00409 static void layer_selector_clear_widgets(LayerSelector *self)
00410 {
00411     GList *list;
00412     GList *temp;
00413
00414     list = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00415     for (temp = list; temp != NULL; temp = temp->next) {
00416         gtk_container_remove(GTK_CONTAINER(self->list_box), GTK_WIDGET(temp->data));
00417     }
00418     /* Widgets are already destroyed when removed from box because they are only referenced inside the
00419     container */
00420
00421     g_list_free(list);
00422
00423     /* Deactivate buttons */
00424     if (self->associated_load_button)
00425         gtk_widget_set_sensitive(self->associated_load_button, FALSE);
00426     if (self->associated_save_button)
00427         gtk_widget_set_sensitive(self->associated_save_button, FALSE);
00428
00435 static gboolean layer_selector_check_if_layer_widget_exists(LayerSelector *self, int layer) {
00436     GList *list;
00437     GList *temp;
00438     LayerElement *widget;
00439     gboolean ret = FALSE;
00440
00441     list = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00442
00443     for (temp = list; temp != NULL; temp = temp->next) {
00444         widget = LAYER_ELEMENT(temp->data);

```

```

00445     if (layer_element_get_layer(widget) == layer) {
00446         ret = TRUE;
00447         break;
00448     }
00449 }
00450
00451 g_list_free(list);
00452
00453 return ret;
00454 }
00455
00461 static void sel_layer_element_setup_dnd_callbacks(LayerSelector *self, LayerElement *element)
00462 {
00463     struct layer_element_dnd_data dnd_data;
00464
00465     if (!self || !element)
00466         return;
00467
00468     dnd_data.entries = &self->dnd_target;
00469     dnd_data.entry_count = 1;
00470     dnd_data.drag_end = sel_layer_element_drag_end;
00471     dnd_data.drag_begin = sel_layer_element_drag_begin;
00472     dnd_data.drag_data_get = sel_layer_element_drag_data_get;
00473
00474     layer_element_set_dnd_callbacks(element, &dnd_data);
00475 }
00476
00482 static void layer_selector_analyze_cell_layers(LayerSelector *self, struct gds_cell *cell)
00483 {
00484     GList *graphics;
00485     struct gds_graphics *gfx;
00486     int layer;
00487     GtkWidget *le;
00488
00489     for (graphics = cell->graphic_objs; graphics != NULL; graphics = graphics->next) {
00490         gfx = (struct gds_graphics *)graphics->data;
00491         layer = (int)gfx->layer;
00492         if (layer_selector_check_if_layer_widget_exists(self, layer) == FALSE) {
00493             le = layer_element_new();
00494             sel_layer_element_setup_dnd_callbacks(self, LAYER_ELEMENT(le));
00495             layer_element_set_layer(LAYER_ELEMENT(le), layer);
00496             gtk_list_box_insert(self->list_box, le, -1);
00497             gtk_widget_show(le);
00498         }
00499     }
00500 }
00501
00510 static gint layer_selector_sort_func(GtkListBoxRow *row1, GtkListBoxRow *row2, gpointer unused)
00511 {
00512     LayerElement *le1, *le2;
00513     gint ret;
00514     static const enum layer_selector_sort_algo default_sort = LAYER_SELECTOR_SORT_DOWN;
00515     const enum layer_selector_sort_algo *algo = (const enum layer_selector_sort_algo *)unused;
00516
00517     /* Assume downward sorting */
00518     /* TODO: This is nasty. Find a better way */
00519     if (!algo)
00520         algo = &default_sort;
00521
00522     le1 = LAYER_ELEMENT(row1);
00523     le2 = LAYER_ELEMENT(row2);
00524
00525     /* Determine sort fow downward sort */
00526     ret = layer_element_get_layer(le1) - layer_element_get_layer(le2);
00527
00528     /* Change order if upward sort is requested */
00529     ret *= (*algo == LAYER_SELECTOR_SORT_DOWN ? 1 : -1);
00530
00531     return ret;
00532 }
00533
00534 void layer_selector_generate_layer_widgets(LayerSelector *selector, GList *libs)
00535 {
00536     GList *cell_list = NULL;
00537     struct gds_library *lib;
00538
00539     layer_selector_clear_widgets(selector);
00540
00541     for (; libs != NULL; libs = libs->next) {
00542         lib = (struct gds_library *)libs->data;
00543         for (cell_list = lib->cells; cell_list != NULL; cell_list = cell_list->next) {
00544             layer_selector_analyze_cell_layers(selector, (struct gds_cell *)cell_list->data);
00545         } /* For Cell List */
00546     } /* For libs */
00547
00548     /* Sort the layers */
00549     layer_selector_force_sort(selector, LAYER_SELECTOR_SORT_DOWN);

```

```

00550
00551     /* Activate Buttons */
00552     if (selector->associated_load_button)
00553         gtk_widget_set_sensitive(selector->associated_load_button, TRUE);
00554     if (selector->associated_save_button)
00555         gtk_widget_set_sensitive(selector->associated_save_button, TRUE);
00556 }
00557
00564 static LayerElement *layer_selector_find_layer_element_in_list(GList *el_list, int layer)
00565 {
00566     LayerElement *ret = NULL;
00567     for (; el_list != NULL; el_list = el_list->next) {
00568         if (layer_element_get_layer(LAYER_ELEMENT(el_list->data)) == layer) {
00569             ret = LAYER_ELEMENT(el_list->data);
00570             break;
00571         }
00572     }
00573     return ret;
00574 }
00575
00587 static void layer_selector_load_layer_mapping_from_file(LayerSelector *self, gchar *file_name)
00588 {
00589     GFile *file;
00590     GFileInputStream *stream;
00591     GDataInputStream *dstream;
00592     LayerElement *le;
00593     char *name;
00594     gboolean export;
00595     int layer;
00596     GdkRGBA color;
00597     int result;
00598     GList *rows;
00599     GList *temp;
00600
00601     file = g_file_new_for_path(file_name);
00602     stream = g_file_read(file, NULL, NULL);
00603
00604     if (!stream)
00605         goto destroy_file;
00606
00607     dstream = g_data_input_stream_new(G_INPUT_STREAM(stream));
00608
00609     rows = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00610
00611     /* Reference and remove all rows from box */
00612     for (temp = rows; temp != NULL; temp = temp->next) {
00613         le = LAYER_ELEMENT(temp->data);
00614         /* Referencing protects the widget from being deleted when removed */
00615         g_object_ref(G_OBJECT(le));
00616         gtk_container_remove(GTK_CONTAINER(self->list_box), GTK_WIDGET(le));
00617     }
00618
00619     while((result = mapping_parser_load_line(dstream, &export, &name, &layer, &color)) >= 0) {
00620         /* skip broken line */
00621         if (result == 1)
00622             continue;
00623
00624         /* Add rows in the same order as in file */
00625         if ((le = layer_selector_find_layer_element_in_list(rows, layer)) {
00626             gtk_list_box_insert(self->list_box, GTK_WIDGET(le), -1);
00627
00628             layer_element_set_color(le, &color);
00629             layer_element_set_export(le, export);
00630             layer_element_set_name(le, name);
00631             g_free(name);
00632
00633             /* Dereference and remove from list */
00634             g_object_unref(G_OBJECT(le));
00635             rows = g_list_remove(rows, le);
00636         }
00637     }
00638
00639     /* Add remaining elements */
00640     for (temp = rows; temp != NULL; temp = temp->next) {
00641         le = LAYER_ELEMENT(temp->data);
00642         /* Referencing protects the widget from being deleted when removed */
00643         gtk_list_box_insert(self->list_box, GTK_WIDGET(le), -1);
00644         g_object_unref(G_OBJECT(le));
00645     }
00646
00647     /* Delete list */
00648     g_list_free(rows);
00649
00650     /* read line */
00651     g_object_unref(dstream);
00652     g_object_unref(stream);
00653     destroy_file;

```

```

00654     g_object_unref(file);
00655 }
00656
00662 static void layer_selector_load_mapping_clicked(GtkWidget *button, gpointer user_data)
00663 {
00664     LayerSelector *sel;
00665     GtkWidget *dialog;
00666     gint res;
00667     gchar *file_name;
00668
00669     sel = LAYER_SELECTOR(user_data);
00670
00671     dialog = gtk_file_chooser_dialog_new("Load Mapping File", GTK_WINDOW(sel->load_parent_window),
GTK_FILE_CHOOSER_ACTION_OPEN,
00672         "Cancel", GTK_RESPONSE_CANCEL, "Load Mapping", GTK_RESPONSE_ACCEPT, NULL);
00673     res = gtk_dialog_run(GTK_DIALOG(dialog));
00674     if (res == GTK_RESPONSE_ACCEPT) {
00675         file_name = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
00676         layer_selector_load_layer_mapping_from_file(sel, file_name);
00677         g_free(file_name);
00678     }
00679     gtk_widget_destroy(dialog);
00680 }
00681
00682
00683
00689 static void layer_selector_save_layer_mapping_data(LayerSelector *self, const gchar *file_name)
00690 {
00691     FILE *file;
00692     char workbuff[512];
00693     GList *le_list;
00694     GList *temp;
00695
00696     /* Overwrite existing file */
00697     file = fopen((const char *)file_name, "w");
00698
00699     le_list = gtk_container_get_children(GTK_CONTAINER(self->list_box));
00700
00701     /* File format is CSV: <Layer>,<target_pos>,<R>,<G>,<B>,<Alpha>,<Export?>,<Name> */
00702     for (temp = le_list; temp != NULL; temp = temp->next) {
00703         /* To be sure it is a valid string */
00704         workbuff[0] = 0;
00705         mapping_parser_gen_csv_line(LAYER_ELEMENT(temp->data), workbuff, sizeof(workbuff));
00706         fwrite(workbuff, sizeof(char), strlen(workbuff), file);
00707     }
00708
00709     g_list_free(le_list);
00710
00711     /* Save File */
00712     fflush(file);
00713     fclose(file);
00714 }
00715
00721 static void layer_selector_save_mapping_clicked(GtkWidget *button, gpointer user_data)
00722 {
00723     GtkWidget *dialog;
00724     gint res;
00725     gchar *file_name;
00726     LayerSelector *sel;
00727
00728     sel = LAYER_SELECTOR(user_data);
00729
00730     dialog = gtk_file_chooser_dialog_new("Save Mapping File", GTK_WINDOW(sel->save_parent_window),
GTK_FILE_CHOOSER_ACTION_SAVE,
00731         "Cancel", GTK_RESPONSE_CANCEL, "Save Mapping", GTK_RESPONSE_ACCEPT, NULL);
00732     gtk_file_chooser_set_do_overwrite_confirmation(GTK_FILE_CHOOSER(dialog), TRUE);
00733
00734     res = gtk_dialog_run(GTK_DIALOG(dialog));
00735     if (res == GTK_RESPONSE_ACCEPT) {
00736         file_name = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
00737         layer_selector_save_layer_mapping_data(sel, file_name);
00738         g_free(file_name);
00739     }
00740     gtk_widget_destroy(dialog);
00741 }
00742
00743 void layer_selector_set_load_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWindow
*main_window)
00744 {
00745     g_clear_object(&selector->load_parent_window);
00746     g_clear_object(&selector->associated_load_button);
00747
00748     g_object_ref(G_OBJECT(button));
00749     g_object_ref(G_OBJECT(main_window));
00750     selector->associated_load_button = button;
00751     selector->load_parent_window = main_window;
00752     g_signal_connect(button, "clicked", G_CALLBACK(layer_selector_load_mapping_clicked), selector);

```

```

00753 }
00754
00755 void layer_selector_set_save_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWidget
*main_window)
00756 {
00757     g_clear_object(&selector->save_parent_window);
00758     g_clear_object(&selector->associated_save_button);
00759
00760     g_object_ref(G_OBJECT(button));
00761     g_object_ref(G_OBJECT(main_window));
00762     selector->associated_save_button = button;
00763     selector->save_parent_window = main_window;
00764     g_signal_connect(button, "clicked", G_CALLBACK(layer_selector_save_mapping_clicked), selector);
00765 }
00766
00767 void layer_selector_force_sort(LayerSelector *selector, enum layer_selector_sort_algo sort_function)
00768 {
00769     GtkWidget *box;
00770
00771     if (!selector)
00772         return;
00773
00774     box = selector->list_box;
00775     if (!box)
00776         return;
00777
00778     /* Set sorting function, sort, and disable sorting function */
00779     gtk_list_box_set_sort_func(box, layer_selector_sort_func, (gpointer)&sort_function, NULL);
00780     gtk_list_box_invalidate_sort(box);
00781     gtk_list_box_set_sort_func(box, NULL, NULL, NULL);
00782 }
00783

```

13.61 layer-selector.dox File Reference

13.62 layer-selector.h File Reference

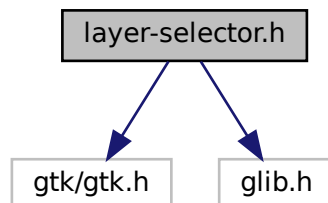
Implementation of the Layer selection list.

```

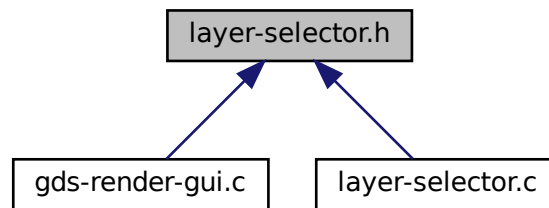
#include <gtk/gtk.h>
#include <glib.h>

```

Include dependency graph for layer-selector.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define` [TYPE_LAYER_SELECTOR](#) ([layer_selector_get_type\(\)](#))

Enumerations

- enum [layer_selector_sort_algo](#) { [LAYER_SELECTOR_SORT_DOWN](#) = 0, [LAYER_SELECTOR_SORT_UP](#) }

Defines how to sort the layer selector list box.

Functions

- `G_BEGIN_DECLS` [G_DECLARE_FINAL_TYPE](#) ([LayerSelector](#), [layer_selector](#), [LAYER_SELECTOR](#), [GObject](#))
- [LayerSelector](#) * [layer_selector_new](#) ([GtkListBox](#) *[list_box](#))
layer_selector_new
- void [layer_selector_generate_layer_widgets](#) ([LayerSelector](#) *[selector](#), [GList](#) *[libs](#))
Generate layer widgets in in the LayerSelector instance.
- void [layer_selector_set_load_mapping_button](#) ([LayerSelector](#) *[selector](#), [GtkWidget](#) *[button](#), [GtkWindow](#) *[main_window](#))
Supply button for loading the layer mapping.
- void [layer_selector_set_save_mapping_button](#) ([LayerSelector](#) *[selector](#), [GtkWidget](#) *[button](#), [GtkWindow](#) *[main_window](#))
Supply button for saving the layer mapping.
- [GList](#) * [layer_selector_export_rendered_layer_info](#) ([LayerSelector](#) *[selector](#))
Get a list of all layers that shall be exported when rendering the cells.
- void [layer_selector_force_sort](#) ([LayerSelector](#) *[selector](#), enum [layer_selector_sort_algo](#) [sort_function](#))
Force the layer selector list to be sorted according to sort_function.

13.62.1 Detailed Description

Implementation of the Layer selection list.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [layer-selector.h](#).

13.63 layer-selector.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef __LAYER_SELECTOR_H__
00032 #define __LAYER_SELECTOR_H__
00033
00034 #include <gtk/gtk.h>
00035 #include <glib.h>
00036
00037 G_BEGIN_DECLS
00038
00039 G_DECLARE_FINAL_TYPE(LayerSelector, layer_selector, LAYER, SELECTOR, GObject);
00040
00041 #define TYPE_LAYER_SELECTOR (layer_selector_get_type())
00042
00046 enum layer_selector_sort_algo {LAYER_SELECTOR_SORT_DOWN = 0, LAYER_SELECTOR_SORT_UP};
00047
00053 LayerSelector *layer_selector_new(GtkListBox *list_box);
00054
00061 void layer_selector_generate_layer_widgets(LayerSelector *selector, GList *libs);
00062
00069 void layer_selector_set_load_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWindow
    *main_window);
00070
00077 void layer_selector_set_save_mapping_button(LayerSelector *selector, GtkWidget *button, GtkWindow
    *main_window);
00078
00084 GList *layer_selector_export_rendered_layer_info(LayerSelector *selector);
00085
00091 void layer_selector_force_sort(LayerSelector *selector, enum layer_selector_sort_algo sort_function);
00092
00093 G_END_DECLS
00094
00095 #endif /* __LAYER_SELECTOR_H__ */
00096

```

13.64 lib-cell-renderer.c File Reference

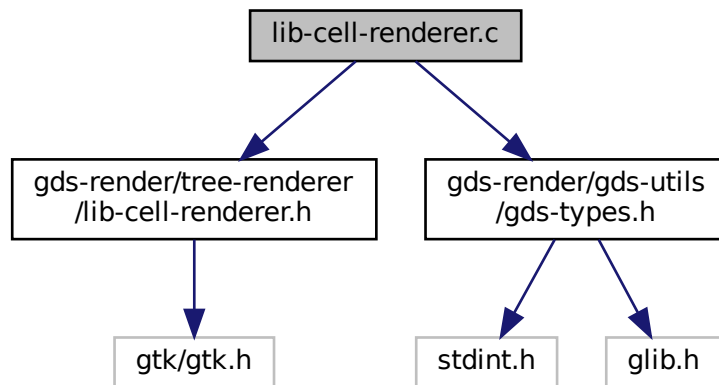
LibCellRenderer GObject Class.

```

#include <gds-render/tree-renderer/lib-cell-renderer.h>
#include <gds-render/gds-utils/gds-types.h>

```

Include dependency graph for lib-cell-renderer.c:



Enumerations

- enum { `PROP_LIB = 1`, `PROP_CELL`, `PROP_ERROR_LEVEL`, `PROP_COUNT` }

Functions

- void `lib_cell_renderer_init` (`LibCellRenderer *self`)
- static void `lib_cell_renderer_constructed` (`GObject *obj`)
- static void `convert_error_level_to_color` (`GdkRGBA *color`, unsigned int `error_level`)
- static void `lib_cell_renderer_set_property` (`GObject *object`, guint `param_id`, const `GValue *value`, `GParamSpec *pspec`)
- static void `lib_cell_renderer_get_property` (`GObject *object`, guint `param_id`, `GValue *value`, `GParamSpec *pspec`)
- void `lib_cell_renderer_class_init` (`LibCellRendererClass *klass`)
- `GtkCellRenderer *` `lib_cell_renderer_new` (void)

Create a new renderer for rendering `gds_cell` and `gds_library` elements.

Variables

- static `GParamSpec *` `properties` [`PROP_COUNT`]

13.64.1 Detailed Description

LibCellRenderer GObject Class.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file `lib-cell-renderer.c`.

13.65 lib-cell-renderer.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <gds-render/tree-renderer/lib-cell-renderer.h>
00032 #include <gds-render/gds-utils/gds-types.h>
00033
00034 G_DEFINE_TYPE(LibCellRenderer, lib_cell_renderer, GTK_TYPE_CELL_RENDERER_TEXT)
00035
00036 enum {
00037     PROP_LIB = 1,
00038     PROP_CELL,
00039     PROP_ERROR_LEVEL,
00040     PROP_COUNT
00041 };
00042
00043 void lib_cell_renderer_init(LibCellRenderer *self)
00044 {
00045     /* Nothing to do */
00046 }
00047
00048 static void lib_cell_renderer_constructed(GObject *obj)
00049 {
00050     G_OBJECT_CLASS(lib_cell_renderer_parent_class)->constructed(obj);
00051 }
00052
00053 static void convert_error_level_to_color(GdkRGBA *color, unsigned int error_level)
00054 {
00055
00056     /* Always use no transparency */
00057     color->alpha = 1.0;
00058
00059     if (error_level & LIB_CELL_RENDERER_ERROR_ERR) {
00060         /* Error set. Color cell red */
00061         color->red = 1.0;
00062         color->blue = 0.0;
00063         color->green = 0.0;
00064     } else if (error_level & LIB_CELL_RENDERER_ERROR_WARN) {
00065         /* Only warning set; orange color */
00066         color->red = 1.0;
00067         color->blue = 0.0;
00068         color->green = 0.6;
00069     } else {
00070         /* Everything okay; green color */
00071         color->red = (double)61.0/(double)255.0;
00072         color->green = (double)152.0/(double)255.0;
00073         color->blue = 0.0;
00074     }
00075 }
00076
00077 static void lib_cell_renderer_set_property(GObject *object,
00078     guint param_id,
00079     const GValue *value,
00080     GParamSpec *pspec)
00081 {
00082     GValue val = G_VALUE_INIT;
00083     GdkRGBA color;
00084
00085     switch (param_id) {
00086     case PROP_LIB:
00087         g_value_init(&val, G_TYPE_STRING);
00088         g_value_set_string(&val, ((struct gds_library *)g_value_get_pointer(value))->name);
00089         g_object_set_property(object, "text", &val);
00090         break;
00091     case PROP_CELL:
00092         g_value_init(&val, G_TYPE_STRING);
00093         g_value_set_string(&val, ((struct gds_cell *)g_value_get_pointer(value))->name);
00094         g_object_set_property(object, "text", &val);
00095         break;
00096     case PROP_ERROR_LEVEL:

```

```

00097     /* Set cell color according to error level */
00098     g_value_init(&val, GDK_TYPE_RGBA);
00099     convert_error_level_to_color(&color, g_value_get_uint(value));
00100     g_value_set_boxed(&val, &color);
00101     g_object_set_property(object, "foreground-rgba", &val);
00102     break;
00103     default:
00104         G_OBJECT_WARN_INVALID_PROPERTY_ID(object, param_id, pspec);
00105         break;
00106     }
00107 }
00108
00109 static void lib_cell_renderer_get_property(GObject      *object,
00110                                           guint         param_id,
00111                                           GValue       *value,
00112                                           GParamSpec   *pspec)
00113 {
00114     switch (param_id) {
00115     default:
00116         G_OBJECT_WARN_INVALID_PROPERTY_ID(object, param_id, pspec);
00117         break;
00118     }
00119 }
00120
00121 static GParamSpec *properties[PROP_COUNT];
00122
00123 void lib_cell_renderer_class_init(LibCellRendererClass *klass)
00124 {
00125     GObjectClass *oclass = G_OBJECT_CLASS(klass);
00126
00127     oclass->constructed = lib_cell_renderer_constructed;
00128     oclass->set_property = lib_cell_renderer_set_property;
00129     oclass->get_property = lib_cell_renderer_get_property;
00130
00131     properties[PROP_LIB] = g_param_spec_pointer("gds-lib", "gds-lib",
00132                                               "Library reference to be displayed",
00133                                               G_PARAM_WRITABLE);
00134     properties[PROP_CELL] = g_param_spec_pointer("gds-cell", "gds-cell",
00135                                               "Cell reference to be displayed",
00136                                               G_PARAM_WRITABLE);
00137     properties[PROP_ERROR_LEVEL] = g_param_spec_uint("error-level", "error-level",
00138                                                    "Error level of this cell", 0, 255, 0, G_PARAM_WRITABLE);
00139
00140     g_object_class_install_properties(oclass, PROP_COUNT, properties);
00141 }
00142
00143 GtkWidget *lib_cell_renderer_new()
00144 {
00145     return GTK_CELL_RENDERER(g_object_new(TYPE_LIB_CELL_RENDERER, NULL));
00146 }
00147

```

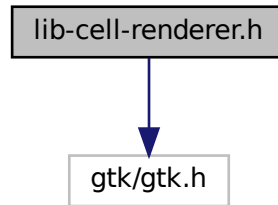
13.66 lib-cell-renderer.dox File Reference

13.67 lib-cell-renderer.h File Reference

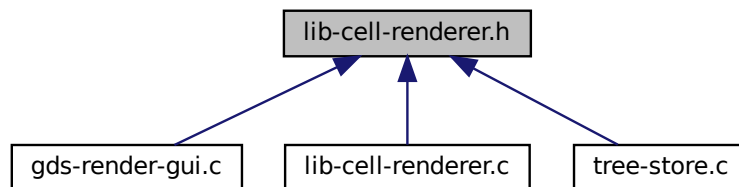
Header file for the LibCellRenderer GObject Class.

```
#include <gtk/gtk.h>
```

Include dependency graph for lib-cell-renderer.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [_LibCellRenderer](#)

Macros

- #define [TYPE_LIB_CELL_RENDERER](#) ([lib_cell_renderer_get_type\(\)](#))
- #define [LIB_CELL_RENDERER_ERROR_WARN](#) (1U<<0)
- #define [LIB_CELL_RENDERER_ERROR_ERR](#) (1U<<1)

Typedefs

- typedef struct [_LibCellRenderer](#) [LibCellRenderer](#)

Functions

- GType [lib_cell_renderer_get_type](#) (void)
lib_cell_renderer_get_type
- GtkCellRenderer * [lib_cell_renderer_new](#) (void)
Create a new renderer for rendering [gds_cell](#) and [gds_library](#) elements.

13.67.1 Detailed Description

Header file for the LibCellRenderer GObject Class.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [lib-cell-renderer.h](#).

13.68 lib-cell-renderer.h

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #ifndef __LIB_CELL_RENDERER_H__
00032 #define __LIB_CELL_RENDERER_H__
00033
00034 #include <gtk/gtk.h>
00035
00036 G_BEGIN_DECLS
00037
00038 G_DECLARE_FINAL_TYPE(LibCellRenderer, lib_cell_renderer, LIB_CELL, RENDERER, GtkCellRendererText)
00039 #define TYPE_LIB_CELL_RENDERER (lib_cell_renderer_get_type())
00040
00044 #define LIB_CELL_RENDERER_ERROR_WARN (1U<<0)
00045 #define LIB_CELL_RENDERER_ERROR_ERR (1U<<1)
00046
00048 typedef struct _LibCellRenderer {
00049     /* Inheritance */
00050     GtkCellRendererText super;
00051     /* Custom Elements */
00052 } LibCellRenderer;
00053
00058 GType lib_cell_renderer_get_type(void);
00059
00064 GtkCellRenderer *lib_cell_renderer_new(void);
00065
00066 G_END_DECLS
00067
00068 #endif /* __LIB_CELL_RENDERER_H__ */
00069
```

13.69 Imf-spec.dox File Reference

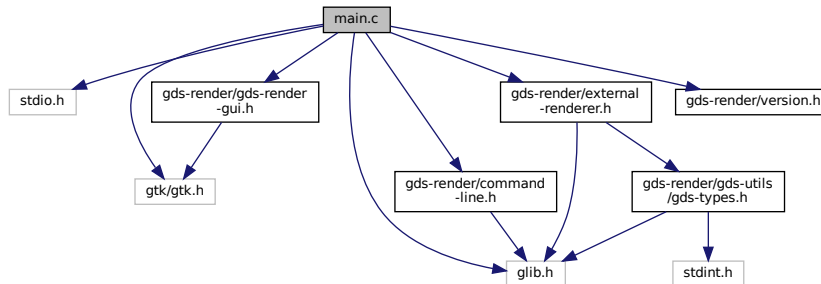
13.70 main-page.dox File Reference

13.71 main.c File Reference

[main.c](#)

```
#include <stdio.h>
#include <gtk/gtk.h>
#include <glib.h>
#include <gds-render/gds-render-gui.h>
#include <gds-render/command-line.h>
#include <gds-render/external-renderer.h>
#include <gds-render/version.h>
```

Include dependency graph for main.c:



Data Structures

- struct [application_data](#)

Structure containing The GtkApplication and a list containing the GdsRenderGui objects.

Functions

- static void [app_quit](#) (GSimpleAction *action, GVariant *parameter, gpointer user_data)
Callback for the menu entry 'Quit'.
- static void [app_about](#) (GSimpleAction *action, GVariant *parameter, gpointer user_data)
Callback for the 'About' menu entry.
- static void [gui_window_closed_callback](#) (GdsRenderGui *gui, gpointer user_data)
Called when a GUI main window is closed.
- static void [gapp_activate](#) (GApplication *app, gpointer user_data)
Activation of the GUI.
- static int [start_gui](#) (int argc, char **argv)
Start the graphical interface.
- static void [print_version](#) (void)
Print the application version string to stdout.
- int [main](#) (int argc, char **argv)
The "entry point" of the application.

Variables

- const static GActionEntry [app_actions](#) []
Contains the application menu entries.

13.71.1 Detailed Description

[main.c](#)

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [main.c](#).

13.71.2 Function Documentation

13.71.2.1 `app_about()`

```
static void app_about (
    GSimpleAction * action,
    GVariant * parameter,
    gpointer user_data ) [static]
```

Callback for the 'About' menu entry.

This function shows the about dialog.

Parameters

<i>action</i>	GSimpleAction, unused
<i>parameter</i>	Unused.
<i>user_data</i>	Unused

Definition at line [83](#) of file [main.c](#).

13.71.2.2 `app_quit()`

```
static void app_quit (
    GSimpleAction * action,
    GVariant * parameter,
    gpointer user_data ) [static]
```

Callback for the menu entry 'Quit'.

Destroys all GUIs contained in the [application_data](#) structure provided by `user_data`.

The complete suspension of all main windows leads to the termination of the GApplication.

Parameters

<i>action</i>	unused
<i>parameter</i>	unused
<i>user_data</i>	application_data structure

Definition at line 56 of file [main.c](#).

13.71.2.3 gapp_activate()

```
static void gapp_activate (
    GApplication * app,
    gpointer user_data ) [static]
```

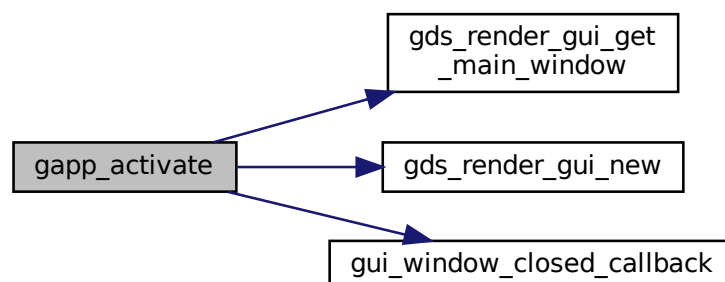
Activation of the GUI.

Parameters

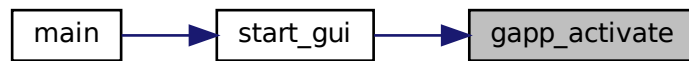
<i>app</i>	The GApplication reference
<i>user_data</i>	Used to store the individual GUI instances.

Definition at line 148 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



13.71.2.4 gui_window_closed_callback()

```

static void gui_window_closed_callback (
    GdsRenderGui * gui,
    gpointer user_data ) [static]
  
```

Called when a GUI main window is closed.

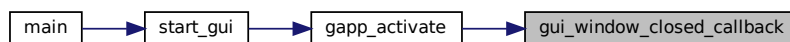
The GdsRenderGui object associated with the closed main window is removed from the list of open GUIs (*user_data*) and unreferenced.

Parameters

<i>gui</i>	The GUI instance the closed main window belongs to
<i>user_data</i>	List of GUIs

Definition at line 134 of file [main.c](#).

Here is the caller graph for this function:



13.71.2.5 main()

```

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

The "entry point" of the application.

Parameters

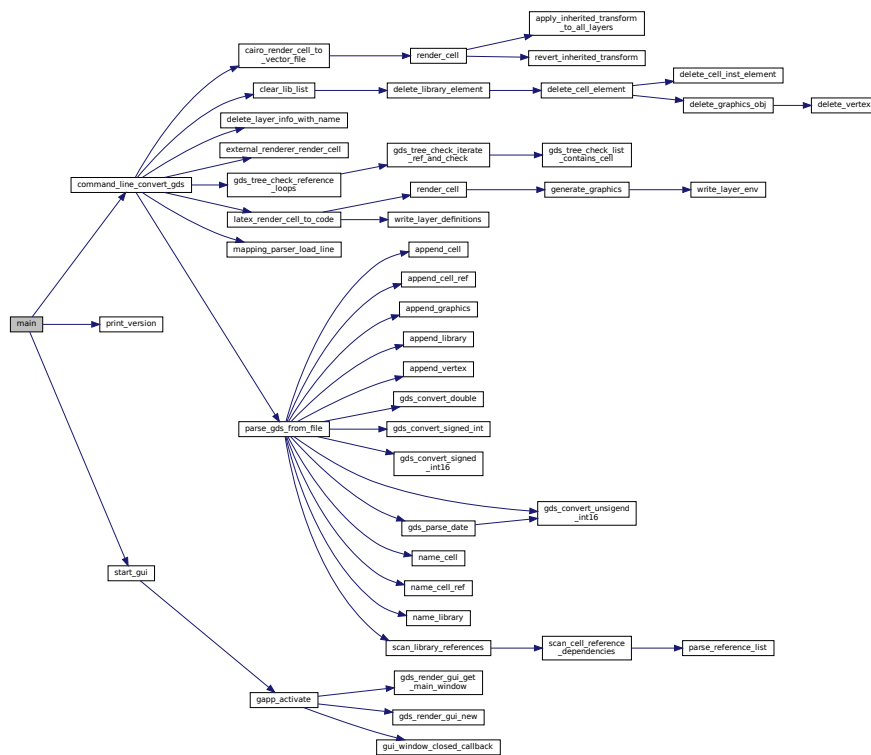
<i>argc</i>	Number of command line parameters
<i>argv</i>	Command line parameters

Returns

Execution status of the application

Definition at line 237 of file [main.c](#).

Here is the call graph for this function:



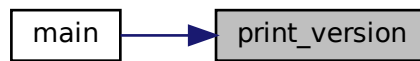
13.71.2.6 print_version()

```
static void print_version (
    void ) [static]
```

Print the application version string to stdout.

Definition at line 225 of file [main.c](#).

Here is the caller graph for this function:



13.71.2.7 start_gui()

```
static int start_gui (  
    int argc,  
    char ** argv ) [static]
```

Start the graphical interface.

This function starts the GUI. If there's already a running instance of this program, a second window will be created in that instance and the second one is terminated.

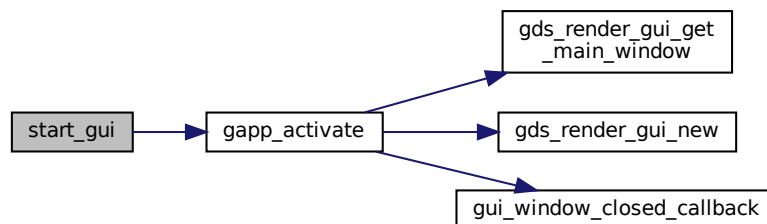
Parameters

<i>argc</i>	
<i>argv</i>	

Returns

Definition at line 177 of file [main.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



13.71.3 Variable Documentation

13.71.3.1 app_actions

```
const static GActionEntry app_actions[] [static]
```

Initial value:

```
= {
    {"quit", app_quit, NULL, NULL, NULL, {0}},
    {"about", app_about, NULL, NULL, NULL, {0}}
}
```

Contains the application menu entries.

Definition at line 120 of file [main.c](#).

13.72 main.c

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #include <stdio.h>
00027 #include <gtk/gtk.h>
00028 #include <glib.h>
00029
00030 #include <gds-render/gds-render-gui.h>
00031 #include <gds-render/command-line.h>
00032 #include <gds-render/external-renderer.h>
00033 #include <gds-render/version.h>
00034
00038 struct application_data {
00039     GtkApplication *app;
00040     GList *gui_list;
00041 };
00042
```

```

00056 static void app_quit(GSimpleAction *action, GVariant *parameter, gpointer user_data)
00057 {
00058     struct application_data * const appdata = (struct application_data *)user_data;
00059     (void)action;
00060     (void)parameter;
00061     GList *list_iter;
00062     GdsRenderGui *gui;
00063
00064     /* Dispose all GUIs */
00065     for (list_iter = appdata->gui_list; list_iter != NULL; list_iter = g_list_next(list_iter)) {
00066         gui = RENDERER_GUI(list_iter->data);
00067         g_object_unref(gui);
00068     }
00069
00070     g_list_free(appdata->gui_list);
00071     appdata->gui_list = NULL;
00072 }
00073
00083 static void app_about(GSimpleAction *action, GVariant *parameter, gpointer user_data)
00084 {
00085     GtkBuilder *builder;
00086     GtkDialog *dialog;
00087     GdkPixbuf *logo_buf;
00088     GError *error = NULL;
00089     (void)user_data;
00090     (void)action;
00091     (void)parameter;
00092
00093     builder = gtk_builder_new_from_resource("/about.glade");
00094     dialog = GTK_DIALOG(gtk_builder_get_object(builder, "about-dialog"));
00095     gtk_window_set_transient_for(GTK_WINDOW(dialog), NULL);
00096     gtk_about_dialog_set_version(GTK_ABOUT_DIALOG(dialog), _app_version_string);
00097
00098     /* Load icon from resource */
00099     logo_buf = gdk_pixbuf_new_from_resource_at_scale("/logo.svg", 100, 100, TRUE, &error);
00100     if (logo_buf) {
00101         /* Set logo */
00102         gtk_about_dialog_set_logo(GTK_ABOUT_DIALOG(dialog), logo_buf);
00103
00104         /* Pixbuf is now owned by about dialog. Unref */
00105         g_object_unref(logo_buf);
00106     } else if (error) {
00107         fprintf(stderr, "Logo could not be displayed: %s\n", error->message);
00108         g_error_free(error);
00109     }
00110
00111     gtk_dialog_run(dialog);
00112
00113     gtk_widget_destroy(GTK_WIDGET(dialog));
00114     g_object_unref(builder);
00115 }
00116
00120 const static GActionEntry app_actions[] = {
00121     {"quit", app_quit, NULL, NULL, NULL, {0}},
00122     {"about", app_about, NULL, NULL, NULL, {0}}
00123 };
00124
00134 static void gui_window_closed_callback(GdsRenderGui *gui, gpointer user_data)
00135 {
00136     GList **gui_list = (GList **)user_data;
00137
00138     /* Dispose of Gui element */
00139     *gui_list = g_list_remove(*gui_list, gui);
00140     g_object_unref(gui);
00141 }
00142
00148 static void gapp_activate(GApplication *app, gpointer user_data)
00149 {
00150     GtkWindow *main_window;
00151     GdsRenderGui *gui;
00152
00153     struct application_data * const appdata = (struct application_data *)user_data;
00154
00155     gui = gds_render_gui_new();
00156     appdata->gui_list = g_list_append(appdata->gui_list, gui);
00157
00158     g_signal_connect(gui, "window-closed", G_CALLBACK(gui_window_closed_callback),
00159                     &appdata->gui_list);
00159
00160     main_window = gds_render_gui_get_main_window(gui);
00161
00162     gtk_application_add_window(GTK_APPLICATION(app), main_window);
00163     gtk_widget_show(GTK_WIDGET(main_window));
00164 }
00165
00177 static int start_gui(int argc, char **argv)
00178 {

```

```

00179
00180     GtkApplication *gapp;
00181     int app_status;
00182     static struct application_data appdata = {
00183         .gui_list = NULL
00184     };
00185     GMenu *menu;
00186     GMenu *m_quit;
00187     GMenu *m_about;
00188
00189     gapp = gtk_application_new("de.shimatta.gds-render", G_APPLICATION_FLAGS_NONE);
00190     g_application_register(G_APPLICATION(gapp), NULL, NULL);
00191     g_signal_connect(gapp, "activate", G_CALLBACK(gapp_activate), &appdata);
00192
00193     if (g_application_get_is_remote(G_APPLICATION(gapp)) == TRUE) {
00194         g_application_activate(G_APPLICATION(gapp));
00195         printf("There is already an open instance. Will open second window in said instance.\n");
00196         return 0;
00197     }
00198
00199     menu = g_menu_new();
00200     m_quit = g_menu_new();
00201     m_about = g_menu_new();
00202     g_menu_append(m_quit, "Quit", "app.quit");
00203     g_menu_append(m_about, "About", "app.about");
00204     g_menu_append_section(menu, NULL, G_MENU_MODEL(m_about));
00205     g_menu_append_section(menu, NULL, G_MENU_MODEL(m_quit));
00206     g_action_map_add_action_entries(G_ACTION_MAP(gapp), app_actions,
00207         G_N_ELEMENTS(app_actions), &appdata);
00208     gtk_application_set_app_menu(GTK_APPLICATION(gapp), G_MENU_MODEL(menu));
00209
00210     g_object_unref(m_quit);
00211     g_object_unref(m_about);
00212     g_object_unref(menu);
00213
00214     app_status = g_application_run(G_APPLICATION(gapp), argc, argv);
00215     g_object_unref(gapp);
00216
00217     g_list_free(appdata.gui_list);
00218
00219     return app_status;
00220 }
00221
00225 static void print_version(void)
00226 {
00227     printf("This is gds-render, version: %s\n\nFor a list of supported commands execute with --help
00228         option.\n",
00229         _app_version_string);
00230 }
00231
00237 int main(int argc, char **argv)
00238 {
00239     int i;
00240     GError *error = NULL;
00241     GOptionContext *context;
00242     gchar *gds_name;
00243     gchar *basename;
00244     gchar *pdfname = NULL, *texname = NULL, *mappingname = NULL, *cellname = NULL, *svgname = NULL;
00245     gboolean tikz = FALSE, pdf = FALSE, pdf_layers = FALSE, pdf_standalone = FALSE, svg = FALSE;
00246     gboolean version = FALSE;
00247     gchar *custom_library_path = NULL;
00248     gchar *custom_library_file_name = NULL;
00249     int scale = 1000;
00250     int app_status = 0;
00251
00252     GOptionEntry entries[] = {
00253         {"version", 'v', 0, G_OPTION_ARG_NONE, &version, "Print version", NULL},
00254         {"tikz", 't', 0, G_OPTION_ARG_NONE, &tikz, "Output TikZ code", NULL },
00255         {"pdf", 'p', 0, G_OPTION_ARG_NONE, &pdf, "Output PDF document", NULL },
00256         /*{"svg", 'S', 0, G_OPTION_ARG_NONE, &svg, "Output SVG image", NULL },
00257         {"scale", 's', 0, G_OPTION_ARG_INT, &scale, "Divide output coordinates by <SCALE>", "<SCALE>"
00258     },
00259     {"tex-output", 'o', 0, G_OPTION_ARG_FILENAME, &texname, "Optional path for TeX file", "PATH"
00260     },
00261     {"pdf-output", 'O', 0, G_OPTION_ARG_FILENAME, &pdfname, "Optional path for PDF file", "PATH"
00262     },
00263     /*{"svg-output", 0, 0, G_OPTION_ARG_FILENAME, &svgname, "Optional path for PDF file", "PATH"},
00264     {"mapping", 'm', 0, G_OPTION_ARG_FILENAME, &mappingname, "Path for Layer Mapping File", "PATH"
00265     },
00266     {"cell", 'c', 0, G_OPTION_ARG_STRING, &cellname, "Cell to render", "NAME" },
00267     {"tex-standalone", 'a', 0, G_OPTION_ARG_NONE, &pdf_standalone, "Create standalone PDF", NULL
00268     },
00269     {"tex-layers", 'l', 0, G_OPTION_ARG_NONE, &pdf_layers, "Create PDF Layers (OCG)", NULL },
00270     {"custom-render-lib", 'P', 0, G_OPTION_ARG_FILENAME, &custom_library_path, "Path to a custom
00271     shared object, that implements the " EXTERNAL_LIBRARY_FUNCTION " function", "PATH"},
00272     {"external-lib-output", 'e', 0, G_OPTION_ARG_FILENAME, &custom_library_file_name, "Output path
00273     for external render library", "PATH"},

```



```

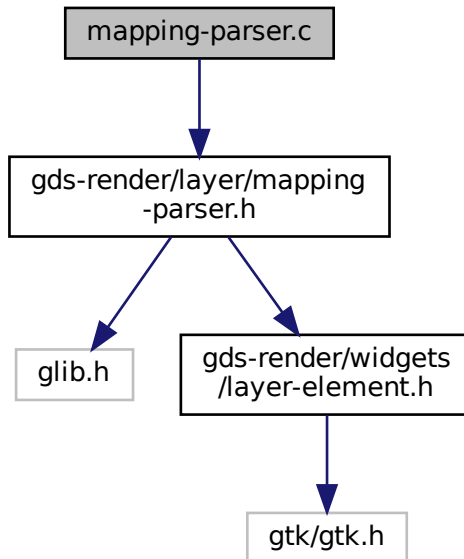
00267     {NULL}
00268 };
00269
00270 context = g_option_context_new(" FILE - Convert GDS file <FILE> to graphic");
00271 g_option_context_add_main_entries(context, entries, NULL);
00272 g_option_context_add_group(context, gtk_get_option_group(TRUE));
00273
00274 if (!g_option_context_parse(context, &argc, &argv, &error)) {
00275     g_print("Option parsing failed: %s\n", error->message);
00276     exit(1);
00277 }
00278
00279 if (version) {
00280     print_version();
00281     goto ret_status;
00282 }
00283
00284 if (argc >= 2) {
00285     if (scale < 1) {
00286         printf("Scale < 1 not allowed. Setting to 1\n");
00287         scale = 1;
00288     }
00289
00290     /* No format selected */
00291     if (!(tikz || pdf || svg))
00292         tikz = TRUE;
00293
00294     /* Get gds name */
00295     gds_name = argv[1];
00296
00297     /* Print out additional arguments as ignored */
00298     for (i = 2; i < argc; i++) {
00299         printf("Ignored argument: %s", argv[i]);
00300     }
00301
00302     /* Check if PDF/TeX names are supplied. if not generate */
00303     basename = g_path_get_basename(gds_name);
00304
00305     if (!texname)
00306         texname = g_strdup_printf("./%s.tex", basename);
00307
00308     if (!pdfname)
00309         pdfname = g_strdup_printf("./%s.pdf", basename);
00310
00311     if (!svgname)
00312         svgname = g_strdup_printf("./%s.svg", basename);
00313
00314     command_line_convert_gds(gds_name, pdfname, texname, pdf, tikz,
00315                             mappingname, cellname, (double)scale,
00316                             pdf_layers, pdf_standalone, svg, svgname,
00317                             custom_library_path, custom_library_file_name);
00318
00319     /* Clean up */
00319     g_free(pdfname);
00320     g_free(texname);
00321     g_free(svgname);
00322     g_free(basename);
00323     if (mappingname)
00324         g_free(mappingname);
00325     if (cellname)
00326         g_free(cellname);
00327     app_status = 0;
00328 } else {
00329     app_status = start_gui(argc, argv);
00330 }
00331
00332 ret_status:
00333     return app_status;
00334 }

```

13.73 mapping-parser.c File Reference

Function to read a mapping file line and parse it.

```
#include <gds-render/layer/mapping-parser.h>
Include dependency graph for mapping-parser.c:
```



Functions

- int [mapping_parser_load_line](#) (GDataInputStream *stream, gboolean *export, char **name, int *layer, GdkRGBA *color)
Load a line from `stream` and parse try to parse it as layer information.
- void [mapping_parser_gen_csv_line](#) (LayerElement *layer_element, char *line_buffer, size_t max_len)
Create Line for LayerMapping file with supplied information.

13.73.1 Detailed Description

Function to read a mapping file line and parse it.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [mapping-parser.c](#).

13.74 mapping-parser.c

```

00001 /*
00002  *
00003  * GDSII-Converter
00004  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00005  *
00006  * This file is part of GDSII-Converter.
00007  *
00008  * GDSII-Converter is free software: you can redistribute it and/or modify
00009  * it under the terms of the GNU General Public License version 2 as
00010  * published by the Free Software Foundation.
00011  *
00012  * GDSII-Converter is distributed in the hope that it will be useful,
00013  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00014  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00015  * GNU General Public License for more details.
00016  *
00017  * You should have received a copy of the GNU General Public License
00018  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00019  */
00020
00032 #include <gds-render/layer/mapping-parser.h>
00033
00034 int mapping_parser_load_line(GDataInputStream *stream, gboolean *export, char **name, int *layer,
00035                             GdkRGBA *color)
00036 {
00037     int ret;
00038     gsize len;
00039     gchar *line;
00040     GRegex *regex;
00041     GMatchInfo *mi;
00042     char *match;
00043     if ((!export) || (!name) || (!layer) || (!color)) {
00044         ret = 1;
00045         goto ret_direct;
00046     }
00047
00048     regex =
00049     g_regex_new("^(<layer>[0-9]+), (?<r>[0-9\\.]+), (?<g>[0-9\\.]+), (?<b>[0-9\\.]+), (?<a>[0-9\\.]+), (?<export>[01]), (?<name>
00050     0, 0, NULL);
00051
00052     line = g_data_input_stream_read_line(stream, &len, NULL, NULL);
00053     if (!line) {
00054         ret = -1;
00055         goto destroy_regex;
00056     }
00057
00058     /* Match line in CSV */
00059     g_regex_match(regex, line, 0, &mi);
00060     if (g_match_info_matches(mi)) {
00061         /* Line is valid */
00062         match = g_match_info_fetch_named(mi, "layer");
00063         *layer = (int)g_ascii_strtoll(match, NULL, 10);
00064         g_free(match);
00065         match = g_match_info_fetch_named(mi, "r");
00066         color->red = g_ascii_strtod(match, NULL);
00067         g_free(match);
00068         match = g_match_info_fetch_named(mi, "g");
00069         color->green = g_ascii_strtod(match, NULL);
00070         g_free(match);
00071         match = g_match_info_fetch_named(mi, "b");
00072         color->blue = g_ascii_strtod(match, NULL);
00073         g_free(match);
00074         match = g_match_info_fetch_named(mi, "a");
00075         color->alpha = g_ascii_strtod(match, NULL);
00076         g_free(match);
00077         match = g_match_info_fetch_named(mi, "export");
00078         *export = (!strcmp(match, "1")) ? TRUE : FALSE;
00079         g_free(match);
00080         match = g_match_info_fetch_named(mi, "name");
00081         *name = match;
00082
00083         ret = 0;
00084     } else {
00085         /* Line is malformed */
00086         printf("Could not recognize line in CSV as valid entry: %s\n", line);
00087         ret = 1;
00088     }
00089
00090     g_match_info_free(mi);
00091     g_free(line);
00092     destroy_regex:
00093     g_regex_unref(regex);
00094     ret_direct:
00095     return ret;

```

```

00094
00095 }
00096
00097 void mapping_parser_gen_csv_line(LayerElement *layer_element, char *line_buffer, size_t max_len)
00098 {
00099     int i;
00100     GString *string;
00101     gboolean export;
00102     const gchar *name;
00103     int layer;
00104     GdkRGBA color;
00105
00106     string = g_string_new_len(NULL, max_len-1);
00107
00108     /* Extract values */
00109     export = layer_element_get_export(layer_element);
00110     name = (const gchar*)layer_element_get_name(layer_element);
00111     layer = layer_element_get_layer(layer_element);
00112     layer_element_get_color(layer_element, &color);
00113
00114     /* print values to line */
00115     g_string_printf(string, "%d:%lf:%lf:%lf:%lf:%d:%s\n",
00116         layer, color.red, color.green,
00117         color.blue, color.alpha, (export == TRUE ? 1 : 0), name);
00118     /* Fix broken locale settings */
00119     for (i = 0; string->str[i]; i++) {
00120         if (string->str[i] == ',')
00121             string->str[i] = '.';
00122     }
00123
00124     for (i = 0; string->str[i]; i++) {
00125         if (string->str[i] == ':')
00126             string->str[i] = ',';
00127     }
00128
00129     if (string->len > (max_len-1)) {
00130         printf("Layer Definition too long. Please shorten Layer Name!!\n");
00131         line_buffer[0] = 0x0;
00132         return;
00133     }
00134
00135     /* copy max_len bytes of string */
00136     strncpy(line_buffer, (char *)string->str, max_len-1);
00137     line_buffer[max_len-1] = 0;
00138
00139     /* Completely remove string */
00140     g_string_free(string, TRUE);
00141 }
00142

```

13.75 mapping-parser.h File Reference

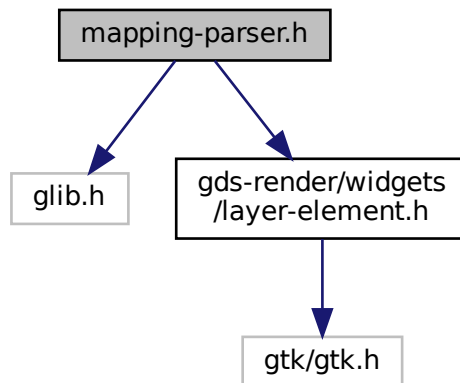
Function to read a mapping file line and parse it.

```

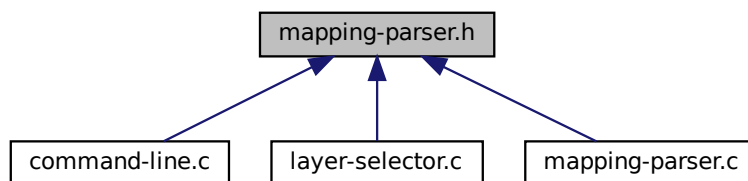
#include <glib.h>
#include <gds-render/widgets/layer-element.h>

```

Include dependency graph for mapping-parser.h:



This graph shows which files directly or indirectly include this file:



Functions

- int [mapping_parser_load_line](#) (GDataInputStream *stream, gboolean *export, char **name, int *layer, GdkRGBA *color)
Load a line from stream and parse try to parse it as layer information.
- void [mapping_parser_gen_csv_line](#) (LayerElement *layer_element, char *line_buffer, size_t max_len)
Create Line for LayerMapping file with supplied information.

13.75.1 Detailed Description

Function to read a mapping file line and parse it.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [mapping-parser.h](#).

13.76 mapping-parser.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 #ifndef __MAPPING_PARSER_H__
00027 #define __MAPPING_PARSER_H__
00028
00034 #include <glib.h>
00035
00036 #include <gds-render/widgets/layer-element.h>
00037
00047 int mapping_parser_load_line(GDataInputStream *stream, gboolean *export, char **name, int *layer,
    GdkRGBA *color);
00048
00055 void mapping_parser_gen_csv_line(LayerElement *layer_element, char *line_buffer, size_t max_len);
00056
00059 #endif /* __MAPPING_PARSER_H__ */

```

13.77 README.MD File Reference

13.78 README.MD

```

00001 # GDS-Render Readme
00002
00003 This software is a rendering programm for GDS2 layout files.
00004 The GDS2 format is mainly used in integrated circuit development.
00005 This program allows the conversion of a GDS file to a vector graphics file.
00006
00007 ## Output Formats
00008 * Export GDS Layout to LaTeX (using TikZ).
00009 * Export to PDF (Cairographics).
00010
00011 # Features
00012 Note: Due to various size limitations of both TikZ and the PDF export, the layout might not render
    correctly. In this case adjust the scale value. A higher scale value scales down your design.
00013
00014 * Configurable layer stack-up.
00015 * Layer colors configurable as ARGB color values.
00016 * Command line interface.
00017 * Awesome Somehow usable GUI.
00018
00019 # License and Other Stuff
00020 * Free software (GPLv2 _only_)
00021 * Coded in plain C using GTK+3.0, Glib2, and Cairographics

```

13.79 renderers.dox File Reference

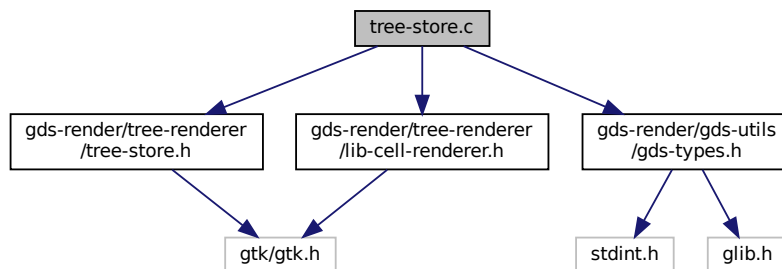
13.80 tree-store.c File Reference

```

#include <gds-render/tree-renderer/tree-store.h>
#include <gds-render/tree-renderer/lib-cell-renderer.h>

```

```
#include <gds-render/gds-utils/gds-types.h>
Include dependency graph for tree-store.c:
```



Functions

- static gboolean `tree_sel_func` (GtkTreeSelection *selection, GtkTreeModel *model, GtkTreePath *path, gboolean path_currently_selected, gpointer data)
this function only allows cells to be selected
- static gboolean `cell_store_filter_visible_func` (GtkTreeModel *model, GtkTreeIter *iter, gpointer data)
cell_store_filter_visible_func Decides whether an element of the tree model model is visible.
- static void `change_filter` (GtkWidget *entry, gpointer data)
- struct `tree_stores` * `setup_cell_selector` (GtkTreeView *view, GtkEntry *search_entry)
Setup a GtkTreeView with the necessary columns.

13.81 tree-store.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020 #include <gds-render/tree-renderer/tree-store.h>
00021 #include <gds-render/tree-renderer/lib-cell-renderer.h>
00022 #include <gds-render/gds-utils/gds-types.h>
00023
00024 static gboolean tree_sel_func(GtkTreeSelection *selection,
00025                               GtkTreeModel *model,
00026                               GtkTreePath *path,
00027                               gboolean path_currently_selected,
00028                               gpointer data)
00029 {
00030     GtkTreeIter iter;
00031     struct gds_cell *cell;
00032     unsigned int error_level;
00033     gboolean ret = FALSE;
00034     (void)selection;
00035     (void)path_currently_selected;

```

```

00056     (void)data;
00057
00058     gtk_tree_model_get_iter(model, &iter, path);
00059     gtk_tree_model_get(model, &iter, CELL_SEL_CELL, &cell, CELL_SEL_CELL_ERROR_STATE, &error_level,
-1);
00060
00061     /* Allow only rows with _valid_ cell to be selected */
00062     if (cell) {
00063         /* Cell available. Check if it passed the critical checks */
00064         if (!(error_level & LIB_CELL_RENDERER_ERROR_ERR))
00065             ret = TRUE;
00066     }
00067
00068     return ret;
00069 }
00070
00071 static gboolean cell_store_filter_visible_func(GtkTreeModel *model, GtkTreeIter *iter, gpointer data)
00072 {
00073     struct tree_stores *stores = (struct tree_stores *)data;
00074     struct gds_cell *cell;
00075     struct gds_library *lib;
00076     gboolean result = FALSE;
00077     const char *search_string;
00078
00079     if (!model || !iter || !stores)
00080         goto exit_filter;
00081
00082     gtk_tree_model_get(model, iter, CELL_SEL_CELL, &cell, CELL_SEL_LIBRARY, &lib, -1);
00083
00084     if (lib) {
00085         result = TRUE;
00086         goto exit_filter;
00087     }
00088
00089     if (!cell)
00090         goto exit_filter;
00091
00092     search_string = gtk_entry_get_text(stores->search_entry);
00093
00094     /* Show all, if field is empty */
00095     if (!strlen(search_string))
00096         result = TRUE;
00097
00098     if (strstr(cell->name, search_string))
00099         result = TRUE;
00100
00101     gtk_tree_view_expand_all(stores->base_tree_view);
00102
00103 exit_filter:
00104     return result;
00105 }
00106
00107 static void change_filter(GtkWidget *entry, gpointer data)
00108 {
00109     struct tree_stores *stores = (struct tree_stores *)data;
00110     (void)entry;
00111
00112     gtk_tree_model_filter_refilter(stores->filter);
00113 }
00114
00115 struct tree_stores *setup_cell_selector(GtkTreeView* view, GtkEntry *search_entry)
00116 {
00117     static struct tree_stores stores;
00118     GtkWidget *render_dates;
00119     GtkWidget *render_cell;
00120     GtkWidget *render_lib;
00121     GtkWidget *column;
00122
00123     stores.base_tree_view = view;
00124     stores.search_entry = search_entry;
00125
00126     stores.base_store = gtk_tree_store_new(CELL_SEL_COLUMN_COUNT, G_TYPE_POINTER, G_TYPE_POINTER,
G_TYPE_UINT, G_TYPE_STRING, G_TYPE_STRING);
00127
00128     /* Searching */
00129     if (search_entry) {
00130         stores.filter =
GTK_TREE_MODEL_FILTER(gtk_tree_model_filter_new(GTK_TREE_MODEL(stores.base_store), NULL));
00131         gtk_tree_model_filter_set_visible_func (stores.filter,
(GtkTreeModelFilterVisibleFunc)cell_store_filter_visible_func,
&stores, NULL);
00132         g_signal_connect (GTK_SEARCH_ENTRY(search_entry), "search-changed", G_CALLBACK(change_filter),
&stores);
00133     }
00134
00135     gtk_tree_view_set_model(view, GTK_TREE_MODEL(stores.filter));
00136 }

```



```

00153     render_dates = gtk_cell_renderer_text_new();
00154     render_cell = lib_cell_renderer_new();
00155     render_lib = lib_cell_renderer_new();
00156
00157     column = gtk_tree_view_column_new_with_attributes("Library", render_lib, "gds-lib",
CELL_SEL_LIBRARY, NULL);
00158     gtk_tree_view_append_column(view, column);
00159
00160     column = gtk_tree_view_column_new_with_attributes("Cell", render_cell, "gds-cell", CELL_SEL_CELL,
00161 "error-level", CELL_SEL_CELL_ERROR_STATE, NULL);
00162     gtk_tree_view_append_column(view, column);
00163
00164     column = gtk_tree_view_column_new_with_attributes("Mod. Date", render_dates, "text",
CELL_SEL_MODDDATE, NULL);
00165     gtk_tree_view_append_column(view, column);
00166
00167     column = gtk_tree_view_column_new_with_attributes("Acc. Date", render_dates, "text",
CELL_SEL_ACCESSDATE, NULL);
00168     gtk_tree_view_append_column(view, column);
00169
00170     /* Callback for selection
00171      * This prevents selecting a library */
00172     gtk_tree_selection_set_select_function(gtk_tree_view_get_selection(view), tree_sel_func, NULL,
NULL);
00173
00174     return &stores;
00175 }

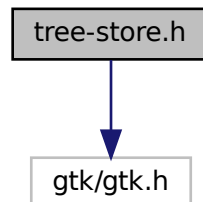
```

13.82 tree-store.h File Reference

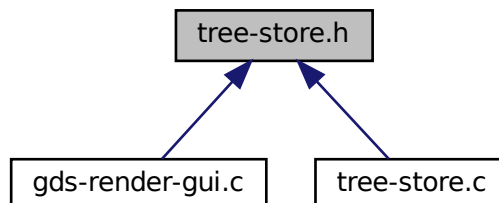
Header file for Tree store implementation.

```
#include <gtk/gtk.h>
```

Include dependency graph for tree-store.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [tree_stores](#)

Enumerations

- enum [cell_store_columns](#) {
[CELL_SEL_LIBRARY](#) = 0, [CELL_SEL_CELL](#), [CELL_SEL_CELL_ERROR_STATE](#), [CELL_SEL_MODDATE](#),
[CELL_SEL_ACCESSDATE](#), [CELL_SEL_COLUMN_COUNT](#) }

Columns of selection tree view.

Functions

- struct [tree_stores](#) * [setup_cell_selector](#) (GtkTreeView *view, GtkEntry *search_entry)

Setup a GtkTreeView with the necessary columns.

13.82.1 Detailed Description

Header file for Tree store implementation.

Tree store implementation.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [tree-store.h](#).

13.83 tree-store.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020 #ifndef __TREE_STORE_H__
00021 #define __TREE_STORE_H__
00022
00023 #include <gtk/gtk.h>
00024
00025 enum cell_store_columns {
00026     CELL_SEL_LIBRARY = 0,
00027     CELL_SEL_CELL,
00028     CELL_SEL_CELL_ERROR_STATE,
00029     CELL_SEL_MODDATE,
00030     CELL_SEL_ACCESSDATE,
00031     CELL_SEL_COLUMN_COUNT
00032 };

```

```

00045
00046 struct tree_stores {
00047     GtkWidget *base_tree_view;
00048     GtkTreeStore *base_store;
00049     GtkTreeModelFilter *filter;
00050     GtkEntry *search_entry;
00051 };
00052
00053 struct tree_stores *setup_cell_selector(GtkTreeView* view, GtkEntry *search_entry);
00054
00055 #endif /* __TREE_STORE_H__ */
00056

```

13.84 usage.dox File Reference

13.85 vector-operations.c File Reference

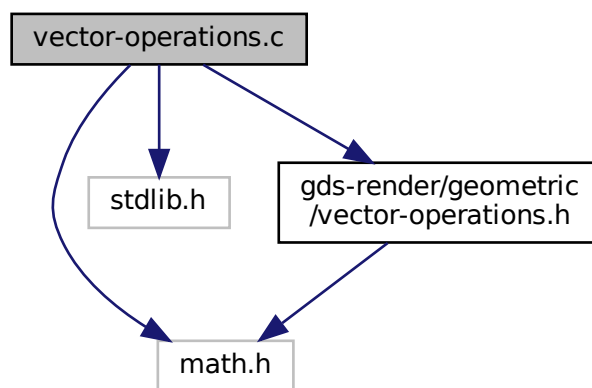
2D Vector operations

```

#include <math.h>
#include <stdlib.h>
#include <gds-render/geometric/vector-operations.h>

```

Include dependency graph for vector-operations.c:



Macros

- #define [ABS_DBL\(a\)](#) ((a) < 0 ? -(a) : (a))

Functions

- double [vector_2d_scalar_multiply](#) (struct [vector_2d](#) *a, struct [vector_2d](#) *b)
- void [vector_2d_normalize](#) (struct [vector_2d](#) *vec)
- void [vector_2d_rotate](#) (struct [vector_2d](#) *vec, double angle)
- struct [vector_2d](#) * [vector_2d_copy](#) (struct [vector_2d](#) *opt_res, struct [vector_2d](#) *vec)

- struct `vector_2d` * `vector_2d_alloc` (void)
- void `vector_2d_free` (struct `vector_2d` *vec)
- void `vector_2d_scale` (struct `vector_2d` *vec, double scale)
- double `vector_2d_abs` (struct `vector_2d` *vec)
- double `vector_2d_calculate_angle_between` (struct `vector_2d` *a, struct `vector_2d` *b)
- void `vector_2d_subtract` (struct `vector_2d` *res, struct `vector_2d` *a, struct `vector_2d` *b)
- void `vector_2d_add` (struct `vector_2d` *res, struct `vector_2d` *a, struct `vector_2d` *b)

13.85.1 Detailed Description

2D Vector operations

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [vector-operations.c](#).

13.86 vector-operations.c

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00031 #include <math.h>
00032 #include <stdlib.h>
00033
00034 #include <gds-render/geometric/vector-operations.h>
00035
00036 #define ABS_DBL(a) ((a) < 0 ? -(a) : (a))
00037
00038 double vector_2d_scalar_multiply(struct vector_2d *a, struct vector_2d *b)
00039 {
00040     if (a && b)
00041         return (a->x * b->x) + (a->y * b->y);
00042     else
00043         return 0.0;
00044 }
00045
00046 void vector_2d_normalize(struct vector_2d *vec)
00047 {
00048     double len;
00049     if (!vec)
00050         return;
00051     len = sqrt(pow(vec->x,2)+pow(vec->y,2));
00052     vec->x = vec->x/len;
00053     vec->y = vec->y/len;
00054 }
00055
00056 void vector_2d_rotate(struct vector_2d *vec, double angle)
00057 {
00058     double sin_val, cos_val;
00059     struct vector_2d temp;
00060
00061     if (!vec)

```

```
00062     return;
00063
00064     sin_val = sin(angle);
00065     cos_val = cos(angle);
00066
00067     vector_2d_copy(&temp, vec);
00068
00069     /* Apply rotation matrix */
00070     vec->x = (cos_val * temp.x) - (sin_val * temp.y);
00071     vec->y = (sin_val * temp.x) + (cos_val * temp.y);
00072 }
00073
00074 struct vector_2d *vector_2d_copy(struct vector_2d *opt_res, struct vector_2d *vec)
00075 {
00076     struct vector_2d *res;
00077
00078     if (!vec)
00079         return NULL;
00080
00081     if (opt_res)
00082         res = opt_res;
00083     else
00084         res = vector_2d_alloc();
00085
00086     if (res) {
00087         res->x = vec->x;
00088         res->y = vec->y;
00089     }
00090     return res;
00091 }
00092
00093 struct vector_2d *vector_2d_alloc(void)
00094 {
00095     return (struct vector_2d *)malloc(sizeof(struct vector_2d));
00096 }
00097
00098 void vector_2d_free(struct vector_2d *vec)
00099 {
00100     if (vec) {
00101         free(vec);
00102     }
00103 }
00104
00105 void vector_2d_scale(struct vector_2d *vec, double scale)
00106 {
00107     if (!vec)
00108         return;
00109
00110     vec->x *= scale;
00111     vec->y *= scale;
00112 }
00113
00114 double vector_2d_abs(struct vector_2d *vec)
00115 {
00116     double len = 0.0;
00117     if (vec) {
00118         len = sqrt(pow(vec->x,2)+pow(vec->y,2));
00119     }
00120     return len;
00121 }
00122
00123 double vector_2d_calculate_angle_between(struct vector_2d *a, struct vector_2d *b)
00124 {
00125     double cos_angle;
00126
00127     if (!a || !b)
00128         return 0.0;
00129
00130     cos_angle = ABS_DBL(vector_2d_scalar_multiply(a, b)) / (vector_2d_abs(a) * vector_2d_abs(b));
00131     return acos(cos_angle);
00132 }
00133
00134 void vector_2d_subtract(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
00135 {
00136     if (res && a && b) {
00137         res->x = a->x - b->x;
00138         res->y = a->y - b->y;
00139     }
00140 }
00141
00142 void vector_2d_add(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b)
00143 {
00144     if (res && a && b) {
00145         res->x = a->x + b->x;
00146         res->y = a->y + b->y;
00147     }
00148 }
```

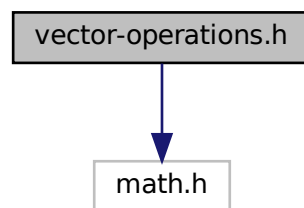
00149

13.87 vector-operations.h File Reference

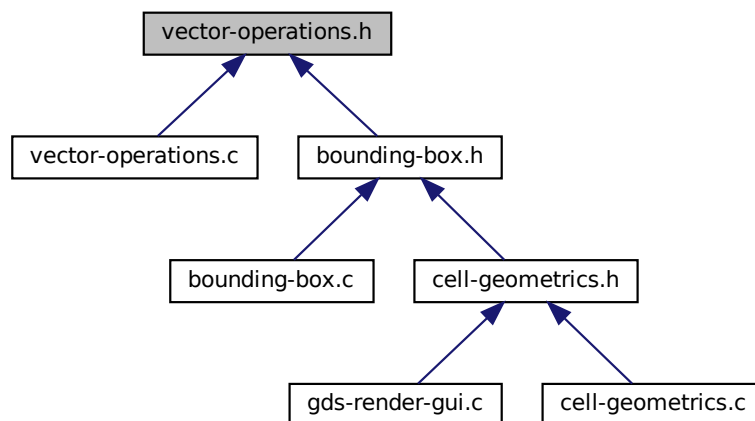
Header for 2D Vector operations.

```
#include <math.h>
```

Include dependency graph for vector-operations.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [vector_2d](#)

Macros

- #define [DEG2RAD\(a\)](#) ((a)*M_PI/180.0)

Functions

- double `vector_2d_scalar_multiply` (struct `vector_2d` *a, struct `vector_2d` *b)
- void `vector_2d_normalize` (struct `vector_2d` *vec)
- void `vector_2d_rotate` (struct `vector_2d` *vec, double angle)
- struct `vector_2d` * `vector_2d_copy` (struct `vector_2d` *opt_res, struct `vector_2d` *vec)
- struct `vector_2d` * `vector_2d_alloc` (void)
- void `vector_2d_free` (struct `vector_2d` *vec)
- void `vector_2d_scale` (struct `vector_2d` *vec, double scale)
- double `vector_2d_abs` (struct `vector_2d` *vec)
- double `vector_2d_calculate_angle_between` (struct `vector_2d` *a, struct `vector_2d` *b)
- void `vector_2d_subtract` (struct `vector_2d` *res, struct `vector_2d` *a, struct `vector_2d` *b)
- void `vector_2d_add` (struct `vector_2d` *res, struct `vector_2d` *a, struct `vector_2d` *b)

13.87.1 Detailed Description

Header for 2D Vector operations.

Author

Mario Hüttel mario.huettel@gmx.net

Definition in file [vector-operations.h](#).

13.88 vector-operations.h

```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00032 #ifndef _VECTOR_OPERATIONS_H_
00033 #define _VECTOR_OPERATIONS_H_
00034
00035 #include <math.h>
00036
00037 struct vector_2d {
00038     double x;
00039     double y;
00040 };
00041
00042 #define DEG2RAD(a) ((a)*M_PI/180.0)
00043
00044 double vector_2d_scalar_multiply(struct vector_2d *a, struct vector_2d *b);
00045 void vector_2d_normalize(struct vector_2d *vec);
00046 void vector_2d_rotate(struct vector_2d *vec, double angle);
00047 struct vector_2d *vector_2d_copy(struct vector_2d *opt_res, struct vector_2d *vec);
00048 struct vector_2d *vector_2d_alloc(void);
00049 void vector_2d_free(struct vector_2d *vec);
00050 void vector_2d_scale(struct vector_2d *vec, double scale);
00051 double vector_2d_abs(struct vector_2d *vec);
00052 double vector_2d_calculate_angle_between(struct vector_2d *a, struct vector_2d *b);
00053 void vector_2d_subtract(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b);
00054 void vector_2d_add(struct vector_2d *res, struct vector_2d *a, struct vector_2d *b);
00055
00056 #endif /* _VECTOR_OPERATIONS_H_ */
00057

```

13.89 version.c File Reference

Variables

- `const char * _app_version_string = "! version not set !"`
This string holds the [Git Based Version Number](#) of the app.

13.90 version.c

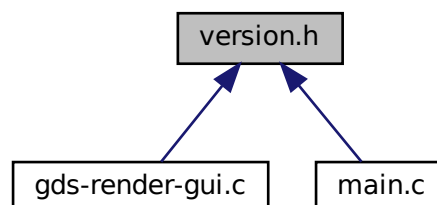
```

00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00020
00028 #ifndef PROJECT_GIT_VERSION
00029 #define xstr(a) str(a)
00030 #define str(a) #a
00031 const char *_app_version_string = xstr(PROJECT_GIT_VERSION);
00032 #else
00033 const char *_app_version_string = "! version not set !";
00034 #endif
00035

```

13.91 version.h File Reference

This graph shows which files directly or indirectly include this file:



Variables

- `const char * _app_version_string`
This string holds the [Git Based Version Number](#) of the app.

13.92 version.h

```
00001 /*
00002  * GDSII-Converter
00003  * Copyright (C) 2018 Mario Hüttel <mario.huettel@gmx.net>
00004  *
00005  * This file is part of GDSII-Converter.
00006  *
00007  * GDSII-Converter is free software: you can redistribute it and/or modify
00008  * it under the terms of the GNU General Public License version 2 as
00009  * published by the Free Software Foundation.
00010  *
00011  * GDSII-Converter is distributed in the hope that it will be useful,
00012  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00013  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00014  * GNU General Public License for more details.
00015  *
00016  * You should have received a copy of the GNU General Public License
00017  * along with GDSII-Converter. If not, see <http://www.gnu.org/licenses/>.
00018  */
00019
00026 extern const char *_app_version_string;
00027
```

13.93 versioning.dox File Reference

13.94 widgets.dox File Reference

Index

- [_GdsRenderGui](#), [152](#)
 - [cell_search_entry](#), [152](#)
 - [cell_tree_store](#), [152](#)
 - [cell_tree_view](#), [153](#)
 - [convert_button](#), [153](#)
 - [gds_libraries](#), [153](#)
 - [layer_selector](#), [153](#)
 - [main_window](#), [153](#)
 - [parent](#), [153](#)
 - [render_dialog_settings](#), [154](#)
- [_LayerElement](#), [154](#)
 - [parent](#), [155](#)
 - [priv](#), [155](#)
- [_LayerElementPriv](#), [155](#)
 - [color](#), [155](#)
 - [event_handle](#), [156](#)
 - [export](#), [156](#)
 - [layer](#), [156](#)
 - [layer_num](#), [156](#)
 - [name](#), [156](#)
- [_LayerSelector](#), [157](#)
 - [associated_load_button](#), [157](#)
 - [associated_save_button](#), [157](#)
 - [dnd_target](#), [157](#)
 - [dummy](#), [157](#)
 - [list_box](#), [158](#)
 - [load_parent_window](#), [158](#)
 - [parent](#), [158](#)
 - [save_parent_window](#), [158](#)
- [_LibCellRenderer](#), [158](#)
 - [super](#), [159](#)
- [_RendererSettingsDialog](#), [159](#)
 - [cell_height](#), [160](#)
 - [cell_width](#), [160](#)
 - [layer_check](#), [160](#)
 - [parent](#), [160](#)
 - [radio_cairo_pdf](#), [160](#)
 - [radio_cairo_svg](#), [160](#)
 - [radio_latex](#), [161](#)
 - [scale](#), [161](#)
 - [shape_drawing](#), [161](#)
 - [standalone_check](#), [161](#)
 - [unit_in_meters](#), [161](#)
 - [x_label](#), [161](#)
 - [x_output_label](#), [162](#)
 - [y_label](#), [162](#)
 - [y_output_label](#), [162](#)
- [_app_version_string](#)
 - [Version Number](#), [129](#)
- [_internal](#)
 - [gds_cell_checks](#), [170](#)
- [ABS_DBL](#)
 - [Geometric Helper Functions](#), [36](#)
- [access_time](#)
 - [gds_cell](#), [168](#)
 - [gds_library](#), [175](#)
- [affected_by_reference_loop](#)
 - [gds_cell_checks](#), [170](#)
- [ANGLE](#)
 - [GDS-Utilities](#), [105](#)
- [angle](#)
 - [gds_cell_instance](#), [172](#)
- [app](#)
 - [application_data](#), [164](#)
- [app_about](#)
 - [main.c](#), [287](#)
- [app_actions](#)
 - [main.c](#), [292](#)
- [app_quit](#)
 - [main.c](#), [287](#)
- [append_cell](#)
 - [GDS-Utilities](#), [106](#)
- [append_cell_ref](#)
 - [GDS-Utilities](#), [107](#)
- [append_graphics](#)
 - [GDS-Utilities](#), [107](#)
- [append_library](#)
 - [GDS-Utilities](#), [108](#)
- [append_vertex](#)
 - [GDS-Utilities](#), [109](#)
- [application_data](#), [163](#)
 - [app](#), [164](#)
 - [gui_list](#), [164](#)
- [apply_inherited_transform_to_all_layers](#)
 - [Cairo Renderer](#), [26](#)
- [associated_load_button](#)
 - [_LayerSelector](#), [157](#)
- [associated_save_button](#)
 - [_LayerSelector](#), [157](#)
- [base_store](#)
 - [tree_stores](#), [184](#)
- [base_tree_view](#)
 - [tree_stores](#), [184](#)
- [BGNLIB](#)
 - [GDS-Utilities](#), [105](#)
- [BGNSTR](#)
 - [GDS-Utilities](#), [105](#)

- BOUNDARY
 - GDS-Utilities, 105
- bounding-box.c, 187, 188
- bounding-box.h, 190, 192
- bounding_box, 164
 - vector_array, 165
 - vectors, 165
- bounding_box::_vectors, 162
 - lower_left, 163
 - upper_right, 163
- bounding_box_apply_transform
 - Geometric Helper Functions, 37
- bounding_box_calculate_path_box
 - Geometric Helper Functions, 38
- bounding_box_calculate_polygon
 - Geometric Helper Functions, 38
- bounding_box_prepare_empty
 - Geometric Helper Functions, 39
- bounding_box_update_box
 - Geometric Helper Functions, 39
- bounding_box_update_point
 - Geometric Helper Functions, 40
- BOX
 - GDS-Utilities, 105
- Cairo Renderer, 25
 - apply_inherited_transform_to_all_layers, 26
 - cairo_render_cell_to_vector_file, 27
 - MAX_LAYERS, 26
 - render_cell, 27
 - revert_inherited_transform, 28
- cairo-output.c, 193, 194
- cairo-output.h, 197, 199
- cairo-renderer.dox, 199
- cairo_layer, 165
 - cr, 166
 - linfo, 166
 - rec, 166
- cairo_render_cell_to_vector_file
 - Cairo Renderer, 27
- calculate_cell_bounding_box
 - Geometric Helper Functions, 40
- calculate_path_miter_points
 - Geometric Helper Functions, 41
- cell-geometrics.c, 199, 201
- cell-geometrics.h, 202, 203
- cell_height
 - _RendererSettingsDialog, 160
- CELL_NAME_MAX
 - GDS-Utilities, 103
- cell_names
 - gds_library, 175
- cell_ref
 - gds_cell_instance, 172
- cell_search_entry
 - _GdsRenderGui, 152
- CELL_SEL_ACCESSDATE
 - Graphical User Interface, 51
- CELL_SEL_CELL
 - Graphical User Interface, 51
- CELL_SEL_CELL_ERROR_STATE
 - Graphical User Interface, 51
- CELL_SEL_COLUMN_COUNT
 - Graphical User Interface, 51
- CELL_SEL_LIBRARY
 - Graphical User Interface, 51
- CELL_SEL_MODDATE
 - Graphical User Interface, 51
- cell_selection_changed
 - Graphical User Interface, 51
- cell_store_columns
 - Graphical User Interface, 51
- cell_store_filter_visible_func
 - Graphical User Interface, 52
- cell_tree_store
 - _GdsRenderGui, 152
- cell_tree_view
 - _GdsRenderGui, 153
- cell_tree_view_activated
 - Graphical User Interface, 53
- cell_width
 - _RendererSettingsDialog, 160
- cells
 - gds_library, 176
- change_filter
 - Graphical User Interface, 54
- checks
 - gds_cell, 168
- child_cells
 - gds_cell, 168
- clear_lib_list
 - GDS-Utilities, 109
- color
 - _LayerElementPriv, 155
 - layer_info, 181
- Command Line Interface, 30
 - command_line_convert_gds, 30
 - delete_layer_info_with_name, 31
- command-line.c, 204
- command-line.dox, 207
- command-line.h, 207, 208
- command_line_convert_gds
 - Command Line Interface, 30
- compilation.dox, 208
- conv-settings-dialog.c, 208, 210
- conv-settings-dialog.h, 215, 216
- conv_generic_to_vector_2d_t
 - Geometric Helper Functions, 37
- convert_button
 - _GdsRenderGui, 153
- convert_error_level_to_color
 - LibCellRenderer GObject, 95
- convert_gds_point_to_2d_vector
 - Geometric Helper Functions, 42
- convert_number_to_engineering
 - RendererSettingsDialog, 132
- cr

- cairo_layer, 166
- Custom GTK Widgets, 100
- datatype
 - gds_graphics, 173
- day
 - gds_time_field, 178
- DEG2RAD
 - Geometric Helper Functions, 36
- delete_cell_element
 - GDS-Utilities, 110
- delete_cell_inst_element
 - GDS-Utilities, 111
- delete_graphics_obj
 - GDS-Utilities, 111
- delete_layer_info_with_name
 - Command Line Interface, 31
- delete_library_element
 - GDS-Utilities, 112
- delete_vertex
 - GDS-Utilities, 113
- dnd_additional_css
 - LayerSelector Object, 92
- dnd_target
 - _LayerSelector, 157
- drag_begin
 - layer_element_dnd_data, 179
- drag_data_get
 - layer_element_dnd_data, 180
- drag_end
 - layer_element_dnd_data, 180
- dummy
 - _LayerSelector, 157
- ENDEL
 - GDS-Utilities, 105
- ENDLIB
 - GDS-Utilities, 105
- ENDSTR
 - GDS-Utilities, 105
- entries
 - layer_element_dnd_data, 180
- entry_count
 - layer_element_dnd_data, 180
- event_handle
 - _LayerElementPriv, 156
- export
 - _LayerElementPriv, 156
- External Shared Object Renderer, 33
 - EXTERNAL_LIBRARY_FUNCTION, 33
 - external_renderer_render_cell, 33
- external-renderer.c, 217
- external-renderer.dox, 219
- external-renderer.h, 219, 221
- EXTERNAL_LIBRARY_FUNCTION
 - External Shared Object Renderer, 33
- external_renderer_render_cell
 - External Shared Object Renderer, 33
- filter
 - tree_stores, 184
- flipped
 - gds_cell_instance, 172
- G_DECLARE_FINAL_TYPE
 - Graphical User Interface, 54
 - LayerSelector Object, 73
- gapp_activate
 - main.c, 288
- gds-parser.c, 221, 224
- gds-parser.h, 233, 235
- gds-render-gui.c, 235
- gds-render-gui.h, 243, 244
- gds-tree-checker.c, 244, 246
- gds-tree-checker.h, 248, 249
- gds-types.h, 249, 251
- GDS-Utilities, 101
 - ANGLE, 105
 - append_cell, 106
 - append_cell_ref, 107
 - append_graphics, 107
 - append_library, 108
 - append_vertex, 109
 - BGNLIB, 105
 - BGNSTR, 105
 - BOUNDARY, 105
 - BOX, 105
 - CELL_NAME_MAX, 103
 - clear_lib_list, 109
 - delete_cell_element, 110
 - delete_cell_inst_element, 111
 - delete_graphics_obj, 111
 - delete_library_element, 112
 - delete_vertex, 113
 - ENDEL, 105
 - ENDLIB, 105
 - ENDSTR, 105
 - GDS_CELL_CHECK_NOT_RUN, 105
 - gds_convert_double, 113
 - gds_convert_signed_int, 114
 - gds_convert_signed_int16, 115
 - gds_convert_unsigend_int16, 115
 - GDS_DEFAULT_UNITS, 103
 - GDS_ERROR, 103
 - GDS_INF, 103
 - gds_parse_date, 116
 - GDS_PRINT_DEBUG_INFOS, 104
 - gds_record, 105
 - gds_tree_check_cell_references, 117
 - gds_tree_check_iterate_ref_and_check, 117
 - gds_tree_check_list_contains_cell, 118
 - gds_tree_check_reference_loops, 119
 - GDS_WARN, 104
 - GRAPHIC_BOX, 106
 - GRAPHIC_PATH, 105
 - GRAPHIC_POLYGON, 106
 - graphics_type, 105
 - HEADER, 105

- INVALID, [105](#)
- LAYER, [105](#)
- LIBNAME, [105](#)
- MAG, [105](#)
- MAX, [104](#)
- MIN, [104](#)
- name_cell, [120](#)
- name_cell_ref, [120](#)
- name_library, [121](#)
- parse_gds_from_file, [121](#)
- parse_reference_list, [122](#)
- PATH, [105](#)
- PATH_FLUSH, [106](#)
- PATH_ROUNDED, [106](#)
- PATH_SQUARED, [106](#)
- path_type, [106](#)
- PATHTYPE, [105](#)
- scan_cell_reference_dependencies, [123](#)
- scan_library_references, [124](#)
- SNAME, [105](#)
- SREF, [105](#)
- STRANS, [105](#)
- STRNAME, [105](#)
- UNITS, [105](#)
- WIDTH, [105](#)
- XY, [105](#)
- gds_cell, [167](#)
 - access_time, [168](#)
 - checks, [168](#)
 - child_cells, [168](#)
 - graphic_objs, [168](#)
 - mod_time, [168](#)
 - name, [168](#)
 - parent_library, [169](#)
- GDS_CELL_CHECK_NOT_RUN
 - GDS-Utilities, [105](#)
- gds_cell_checks, [169](#)
 - _internal, [170](#)
 - affected_by_reference_loop, [170](#)
 - unresolved_child_count, [170](#)
- gds_cell_checks::check_internals, [151](#)
 - marker, [151](#)
- gds_cell_instance, [171](#)
 - angle, [172](#)
 - cell_ref, [172](#)
 - flipped, [172](#)
 - magnification, [172](#)
 - origin, [172](#)
 - ref_name, [172](#)
- gds_convert_double
 - GDS-Utilities, [113](#)
- gds_convert_signed_int
 - GDS-Utilities, [114](#)
- gds_convert_signed_int16
 - GDS-Utilities, [115](#)
- gds_convert_unsigned_int16
 - GDS-Utilities, [115](#)
- GDS_DEFAULT_UNITS
 - GDS-Utilities, [103](#)
- GDS_ERROR
 - GDS-Utilities, [103](#)
- gds_graphics, [173](#)
 - datatype, [173](#)
 - gfx_type, [173](#)
 - layer, [174](#)
 - path_render_type, [174](#)
 - vertices, [174](#)
 - width_absolute, [174](#)
- GDS_INF
 - GDS-Utilities, [103](#)
- gds_libraries
 - _GdsRenderGui, [153](#)
- gds_library, [175](#)
 - access_time, [175](#)
 - cell_names, [175](#)
 - cells, [176](#)
 - mod_time, [176](#)
 - name, [176](#)
 - unit_in_meters, [176](#)
- gds_parse_date
 - GDS-Utilities, [116](#)
- gds_point, [177](#)
 - x, [177](#)
 - y, [177](#)
- GDS_PRINT_DEBUG_INFOS
 - GDS-Utilities, [104](#)
- gds_record
 - GDS-Utilities, [105](#)
- gds_render_gui_class_init
 - Graphical User Interface, [54](#)
- gds_render_gui_dispose
 - Graphical User Interface, [54](#)
- gds_render_gui_get_main_window
 - Graphical User Interface, [55](#)
- gds_render_gui_init
 - Graphical User Interface, [55](#)
- gds_render_gui_new
 - Graphical User Interface, [56](#)
- gds_render_gui_signal_sig_ids
 - Graphical User Interface, [51](#)
- gds_render_gui_signals
 - Graphical User Interface, [63](#)
- gds_time_field, [177](#)
 - day, [178](#)
 - hour, [178](#)
 - minute, [178](#)
 - month, [178](#)
 - second, [178](#)
 - year, [179](#)
- gds_tree_check_cell_references
 - GDS-Utilities, [117](#)
- gds_tree_check_iterate_ref_and_check
 - GDS-Utilities, [117](#)
- gds_tree_check_list_contains_cell
 - GDS-Utilities, [118](#)
- gds_tree_check_reference_loops

- GDS-Utilities, [119](#)
- GDS_WARN
 - GDS-Utilities, [104](#)
- generate_graphics
 - LaTeX/TikZ Renderer, [65](#)
- generate_string_from_date
 - Graphical User Interface, [57](#)
- Geometric Helper Functions, [35](#)
 - ABS_DBL, [36](#)
 - bounding_box_apply_transform, [37](#)
 - bounding_box_calculate_path_box, [38](#)
 - bounding_box_calculate_polygon, [38](#)
 - bounding_box_prepare_empty, [39](#)
 - bounding_box_update_box, [39](#)
 - bounding_box_update_point, [40](#)
 - calculate_cell_bounding_box, [40](#)
 - calculate_path_miter_points, [41](#)
 - conv_generic_to_vector_2d_t, [37](#)
 - convert_gds_point_to_2d_vector, [42](#)
 - DEG2RAD, [36](#)
 - MAX, [36](#)
 - MIN, [36](#)
 - update_box_with_gfx, [42](#)
 - vector_2d_abs, [43](#)
 - vector_2d_add, [43](#)
 - vector_2d_alloc, [44](#)
 - vector_2d_calculate_angle_between, [44](#)
 - vector_2d_copy, [45](#)
 - vector_2d_free, [46](#)
 - vector_2d_normalize, [46](#)
 - vector_2d_rotate, [46](#)
 - vector_2d_scalar_multiply, [47](#)
 - vector_2d_scale, [47](#)
 - vector_2d_subtract, [48](#)
- geometric.dox, [252](#)
- gfx_type
 - gds_graphics, [173](#)
- gpl-2.0.md, [252](#)
- GRAPHIC_BOX
 - GDS-Utilities, [106](#)
- graphic_objs
 - gds_cell, [168](#)
- GRAPHIC_PATH
 - GDS-Utilities, [105](#)
- GRAPHIC_POLYGON
 - GDS-Utilities, [106](#)
- Graphical User Interface, [49](#)
 - CELL_SEL_ACCESSDATE, [51](#)
 - CELL_SEL_CELL, [51](#)
 - CELL_SEL_CELL_ERROR_STATE, [51](#)
 - CELL_SEL_COLUMN_COUNT, [51](#)
 - CELL_SEL_LIBRARY, [51](#)
 - CELL_SEL_MODDATE, [51](#)
 - cell_selection_changed, [51](#)
 - cell_store_columns, [51](#)
 - cell_store_filter_visible_func, [52](#)
 - cell_tree_view_activated, [53](#)
 - change_filter, [54](#)
- G_DECLARE_FINAL_TYPE, [54](#)
- gds_render_gui_class_init, [54](#)
- gds_render_gui_dispose, [54](#)
- gds_render_gui_get_main_window, [55](#)
- gds_render_gui_init, [55](#)
- gds_render_gui_new, [56](#)
- gds_render_gui_signal_sig_ids, [51](#)
- gds_render_gui_signals, [63](#)
- generate_string_from_date, [57](#)
- on_convert_clicked, [57](#)
- on_load_gds, [58](#)
- on_window_close, [59](#)
- RENDERER_TYPE_GUI, [50](#)
- setup_cell_selector, [60](#)
- SIGNAL_COUNT, [51](#)
- SIGNAL_WINDOW_CLOSED, [51](#)
- sort_down_callback, [61](#)
- sort_up_callback, [62](#)
- tree_sel_func, [62](#)
- graphics_type
 - GDS-Utilities, [105](#)
- gui.dox, [252](#)
- gui_list
 - application_data, [164](#)
- gui_window_closed_callback
 - main.c, [289](#)
- HEADER
 - GDS-Utilities, [105](#)
- hide_tex_options
 - RendererSettingsDialog, [132](#)
- hour
 - gds_time_field, [178](#)
- INVALID
 - GDS-Utilities, [105](#)
- latex-output.c, [252](#), [253](#)
- latex-output.h, [256](#), [257](#)
- latex-renderer.dox, [258](#)
- LaTeX/TikZ Renderer, [64](#)
 - generate_graphics, [65](#)
 - LATEX_LINE_BUFFER_KB, [64](#)
 - latex_render_cell_to_code, [66](#)
 - render_cell, [67](#)
 - write_layer_definitions, [67](#)
 - write_layer_env, [69](#)
 - WRITEOUT_BUFFER, [64](#)
- LATEX_LINE_BUFFER_KB
 - LaTeX/TikZ Renderer, [64](#)
- latex_render_callback
 - RendererSettingsDialog, [132](#)
- latex_render_cell_to_code
 - LaTeX/TikZ Renderer, [66](#)
- LAYER
 - GDS-Utilities, [105](#)
- layer
 - _LayerElementPriv, [156](#)
 - gds_graphics, [174](#)

- layer_info, 181
- layer-element.c, 258, 259
- layer-element.h, 261, 263
- layer-info.c, 264, 265
 - layer_info_delete_struct, 264
- layer-info.h, 265, 267
 - layer_info_delete_struct, 267
- layer-selector.c, 268, 270
- layer-selector.dox, 278
- layer-selector.h, 278, 280
- layer_check
 - _RendererSettingsDialog, 160
- layer_element_class_init
 - LayerElement, 143
- layer_element_constructed
 - LayerElement, 143
- layer_element_dispose
 - LayerElement, 144
- layer_element_dnd_data, 179
 - drag_begin, 179
 - drag_data_get, 180
 - drag_end, 180
 - entries, 180
 - entry_count, 180
- layer_element_get_color
 - LayerElement, 144
- layer_element_get_export
 - LayerElement, 145
- layer_element_get_layer
 - LayerElement, 145
- layer_element_get_name
 - LayerElement, 146
- layer_element_init
 - LayerElement, 146
- layer_element_new
 - LayerElement, 147
- layer_element_set_color
 - LayerElement, 147
- layer_element_set_dnd_callbacks
 - LayerElement, 148
- layer_element_set_export
 - LayerElement, 148
- layer_element_set_layer
 - LayerElement, 149
- layer_element_set_name
 - LayerElement, 149
- layer_info, 181
 - color, 181
 - layer, 181
 - name, 181
 - stacked_position, 182
- layer_info_delete_struct
 - layer-info.c, 264
 - layer-info.h, 267
- layer_num
 - _LayerElementPriv, 156
- layer_selector
 - _GdsRenderGui, 153
- layer_selector_analyze_cell_layers
 - LayerSelector Object, 73
- layer_selector_check_if_layer_widget_exists
 - LayerSelector Object, 74
- layer_selector_class_init
 - LayerSelector Object, 75
- layer_selector_clear_widgets
 - LayerSelector Object, 75
- layer_selector_dispose
 - LayerSelector Object, 76
- layer_selector_drag_data_received
 - LayerSelector Object, 76
- layer_selector_drag_leave
 - LayerSelector Object, 76
- layer_selector_drag_motion
 - LayerSelector Object, 77
- layer_selector_export_rendered_layer_info
 - LayerSelector Object, 78
- layer_selector_find_layer_element_in_list
 - LayerSelector Object, 79
- layer_selector_force_sort
 - LayerSelector Object, 79
- layer_selector_generate_layer_widgets
 - LayerSelector Object, 80
- layer_selector_get_last_row
 - LayerSelector Object, 81
- layer_selector_get_row_after
 - LayerSelector Object, 81
- layer_selector_get_row_before
 - LayerSelector Object, 82
- layer_selector_init
 - LayerSelector Object, 82
- layer_selector_load_layer_mapping_from_file
 - LayerSelector Object, 82
- layer_selector_load_mapping_clicked
 - LayerSelector Object, 83
- layer_selector_new
 - LayerSelector Object, 84
- layer_selector_save_layer_mapping_data
 - LayerSelector Object, 85
- layer_selector_save_mapping_clicked
 - LayerSelector Object, 86
- layer_selector_set_load_mapping_button
 - LayerSelector Object, 87
- layer_selector_set_save_mapping_button
 - LayerSelector Object, 88
- layer_selector_setup_dnd
 - LayerSelector Object, 88
- layer_selector_sort_algo
 - LayerSelector Object, 73
- LAYER_SELECTOR_SORT_DOWN
 - LayerSelector Object, 73
- layer_selector_sort_func
 - LayerSelector Object, 89
- LAYER_SELECTOR_SORT_UP
 - LayerSelector Object, 73
- LayerElement, 142
 - layer_element_class_init, 143

- layer_element_constructed, [143](#)
- layer_element_dispose, [144](#)
- layer_element_get_color, [144](#)
- layer_element_get_export, [145](#)
- layer_element_get_layer, [145](#)
- layer_element_get_name, [146](#)
- layer_element_init, [146](#)
- layer_element_new, [147](#)
- layer_element_set_color, [147](#)
- layer_element_set_dnd_callbacks, [148](#)
- layer_element_set_export, [148](#)
- layer_element_set_layer, [149](#)
- layer_element_set_name, [149](#)
- LayerElementPriv, [143](#)
- TYPE_LAYER_ELEMENT, [143](#)
- LayerElementPriv
 - LayerElement, [143](#)
- LayerSelector Object, [71](#)
 - dnd_additional_css, [92](#)
 - G_DECLARE_FINAL_TYPE, [73](#)
 - layer_selector_analyze_cell_layers, [73](#)
 - layer_selector_check_if_layer_widget_exists, [74](#)
 - layer_selector_class_init, [75](#)
 - layer_selector_clear_widgets, [75](#)
 - layer_selector_dispose, [76](#)
 - layer_selector_drag_data_received, [76](#)
 - layer_selector_drag_leave, [76](#)
 - layer_selector_drag_motion, [77](#)
 - layer_selector_export_rendered_layer_info, [78](#)
 - layer_selector_find_layer_element_in_list, [79](#)
 - layer_selector_force_sort, [79](#)
 - layer_selector_generate_layer_widgets, [80](#)
 - layer_selector_get_last_row, [81](#)
 - layer_selector_get_row_after, [81](#)
 - layer_selector_get_row_before, [82](#)
 - layer_selector_init, [82](#)
 - layer_selector_load_layer_mapping_from_file, [82](#)
 - layer_selector_load_mapping_clicked, [83](#)
 - layer_selector_new, [84](#)
 - layer_selector_save_layer_mapping_data, [85](#)
 - layer_selector_save_mapping_clicked, [86](#)
 - layer_selector_set_load_mapping_button, [87](#)
 - layer_selector_set_save_mapping_button, [88](#)
 - layer_selector_setup_dnd, [88](#)
 - layer_selector_sort_algo, [73](#)
 - LAYER_SELECTOR_SORT_DOWN, [73](#)
 - layer_selector_sort_func, [89](#)
 - LAYER_SELECTOR_SORT_UP, [73](#)
 - sel_layer_element_drag_begin, [90](#)
 - sel_layer_element_drag_data_get, [90](#)
 - sel_layer_element_drag_end, [91](#)
 - sel_layer_element_setup_dnd_callbacks, [91](#)
 - TYPE_LAYER_SELECTOR, [72](#)
- lib-cell-renderer.c, [280](#), [282](#)
- lib-cell-renderer.dox, [283](#)
- lib-cell-renderer.h, [283](#), [285](#)
- lib_cell_renderer_class_init
 - LibCellRenderer GObject, [95](#)
- lib_cell_renderer_constructed
 - LibCellRenderer GObject, [96](#)
- LIB_CELL_RENDERER_ERROR_ERR
 - LibCellRenderer GObject, [94](#)
- LIB_CELL_RENDERER_ERROR_WARN
 - LibCellRenderer GObject, [94](#)
- lib_cell_renderer_get_property
 - LibCellRenderer GObject, [96](#)
- lib_cell_renderer_get_type
 - LibCellRenderer GObject, [97](#)
- lib_cell_renderer_init
 - LibCellRenderer GObject, [97](#)
- lib_cell_renderer_new
 - LibCellRenderer GObject, [97](#)
- lib_cell_renderer_set_property
 - LibCellRenderer GObject, [97](#)
- LibCellRenderer
 - LibCellRenderer GObject, [94](#)
- LibCellRenderer GObject, [93](#)
 - convert_error_level_to_color, [95](#)
 - lib_cell_renderer_class_init, [95](#)
 - lib_cell_renderer_constructed, [96](#)
 - LIB_CELL_RENDERER_ERROR_ERR, [94](#)
 - LIB_CELL_RENDERER_ERROR_WARN, [94](#)
 - lib_cell_renderer_get_property, [96](#)
 - lib_cell_renderer_get_type, [97](#)
 - lib_cell_renderer_init, [97](#)
 - lib_cell_renderer_new, [97](#)
 - lib_cell_renderer_set_property, [97](#)
 - LibCellRenderer, [94](#)
 - PROP_CELL, [95](#)
 - PROP_COUNT, [95](#)
 - PROP_ERROR_LEVEL, [95](#)
 - PROP_LIB, [95](#)
 - properties, [98](#)
 - TYPE_LIB_CELL_RENDERER, [94](#)
- LIBNAME
 - GDS-Utilities, [105](#)
- linfo
 - cairo_layer, [166](#)
- list_box
 - _LayerSelector, [158](#)
- lmf-spec.dox, [285](#)
- load_parent_window
 - _LayerSelector, [158](#)
- lower_left
 - bounding_box::_vectors, [163](#)
- MAG
 - GDS-Utilities, [105](#)
- magnification
 - gds_cell_instance, [172](#)
- main
 - main.c, [289](#)
- main-page.dox, [285](#)
- main.c, [285](#)
 - app_about, [287](#)
 - app_actions, [292](#)
 - app_quit, [287](#)

- gapp_activate, 288
- gui_window_closed_callback, 289
- main, 289
- print_version, 290
- start_gui, 291
- main_window
 - _GdsRenderGui, 153
- Mapping-Parser, 126
 - mapping_parser_gen_csv_line, 126
 - mapping_parser_load_line, 127
- mapping-parser.c, 295, 297
- mapping-parser.h, 298, 300
- mapping_parser_gen_csv_line
 - Mapping-Parser, 126
- mapping_parser_load_line
 - Mapping-Parser, 127
- marker
 - gds_cell_checks::_check_internals, 151
- MAX
 - GDS-Utilities, 104
 - Geometric Helper Functions, 36
- MAX_LAYERS
 - Cairo Renderer, 26
- MIN
 - GDS-Utilities, 104
 - Geometric Helper Functions, 36
- minute
 - gds_time_field, 178
- mod_time
 - gds_cell, 168
 - gds_library, 176
- month
 - gds_time_field, 178
- name
 - _LayerElementPriv, 156
 - gds_cell, 168
 - gds_library, 176
 - layer_info, 181
- name_cell
 - GDS-Utilities, 120
- name_cell_ref
 - GDS-Utilities, 120
- name_library
 - GDS-Utilities, 121
- on_convert_clicked
 - Graphical User Interface, 57
- on_load_gds
 - Graphical User Interface, 58
- on_window_close
 - Graphical User Interface, 59
- origin
 - gds_cell_instance, 172
- Output Renderers, 99
- output_renderer
 - RendererSettingsDialog, 131
- parent
 - _GdsRenderGui, 153
 - _LayerElement, 155
 - _LayerSelector, 158
 - _RenderSettingsDialog, 160
- parent_library
 - gds_cell, 169
- parse_gds_from_file
 - GDS-Utilities, 121
- parse_reference_list
 - GDS-Utilities, 122
- PATH
 - GDS-Utilities, 105
- PATH_FLUSH
 - GDS-Utilities, 106
- path_render_type
 - gds_graphics, 174
- PATH_ROUNDED
 - GDS-Utilities, 106
- PATH_SQUARED
 - GDS-Utilities, 106
- path_type
 - GDS-Utilities, 106
- PATHTYPE
 - GDS-Utilities, 105
- print_version
 - main.c, 290
- priv
 - _LayerElement, 155
- PROP_CELL
 - LibCellRenderer GObject, 95
- PROP_CELL_NAME
 - RendererSettingsDialog, 131
- PROP_COUNT
 - LibCellRenderer GObject, 95
 - RendererSettingsDialog, 131
- PROP_ERROR_LEVEL
 - LibCellRenderer GObject, 95
- PROP_LIB
 - LibCellRenderer GObject, 95
- properties
 - LibCellRenderer GObject, 98
 - RendererSettingsDialog, 141
- radio_cairo_pdf
 - _RenderSettingsDialog, 160
- radio_cairo_svg
 - _RenderSettingsDialog, 160
- radio_latex
 - _RenderSettingsDialog, 161
- README.MD, 300
- rec
 - cairo_layer, 166
- ref_name
 - gds_cell_instance, 172
- render_cell
 - Cairo Renderer, 27
 - LaTeX/TikZ Renderer, 67
- render_dialog_settings
 - _GdsRenderGui, 154

- render_settings, 182
 - renderer, 183
 - scale, 183
 - tex_pdf_layers, 183
 - tex_standalone, 183
- renderer
 - render_settings, 183
- RENDERER_CAIROGRAPHICS_PDF
 - RendererSettingsDialog, 132
- RENDERER_CAIROGRAPHICS_SVG
 - RendererSettingsDialog, 132
- RENDERER_LATEX_TIKZ
 - RendererSettingsDialog, 132
- renderer_settings_dialog_class_init
 - RendererSettingsDialog, 133
- renderer_settings_dialog_get_property
 - RendererSettingsDialog, 134
- renderer_settings_dialog_get_settings
 - RendererSettingsDialog, 134
- renderer_settings_dialog_init
 - RendererSettingsDialog, 134
- renderer_settings_dialog_new
 - RendererSettingsDialog, 135
- renderer_settings_dialog_set_cell_height
 - RendererSettingsDialog, 135
- renderer_settings_dialog_set_cell_width
 - RendererSettingsDialog, 136
- renderer_settings_dialog_set_database_unit_scale
 - RendererSettingsDialog, 137
- renderer_settings_dialog_set_property
 - RendererSettingsDialog, 138
- renderer_settings_dialog_set_settings
 - RendererSettingsDialog, 138
- renderer_settings_dialog_update_labels
 - RendererSettingsDialog, 139
- RENDERER_TYPE_GUI
 - Graphical User Interface, 50
- RENDERER_TYPE_SETTINGS_DIALOG
 - RendererSettingsDialog, 131
- renderers.dox, 300
- RendererSettingsDialog, 130
 - convert_number_to_engineering, 132
 - hide_tex_options, 132
 - latex_render_callback, 132
 - output_renderer, 131
 - PROP_CELL_NAME, 131
 - PROP_COUNT, 131
 - properties, 141
 - RENDERER_CAIROGRAPHICS_PDF, 132
 - RENDERER_CAIROGRAPHICS_SVG, 132
 - RENDERER_LATEX_TIKZ, 132
 - renderer_settings_dialog_class_init, 133
 - renderer_settings_dialog_get_property, 134
 - renderer_settings_dialog_get_settings, 134
 - renderer_settings_dialog_init, 134
 - renderer_settings_dialog_new, 135
 - renderer_settings_dialog_set_cell_height, 135
 - renderer_settings_dialog_set_cell_width, 136
 - renderer_settings_dialog_set_database_unit_scale, 137
 - renderer_settings_dialog_set_property, 138
 - renderer_settings_dialog_set_settings, 138
 - renderer_settings_dialog_update_labels, 139
 - RENDERER_TYPE_SETTINGS_DIALOG, 131
 - scale_value_changed, 139
 - shape_drawer_drawing_callback, 140
 - show_tex_options, 140
- revert_inherited_transform
 - Cairo Renderer, 28
- save_parent_window
 - _LayerSelector, 158
- scale
 - _RendererSettingsDialog, 161
 - render_settings, 183
- scale_value_changed
 - RendererSettingsDialog, 139
- scan_cell_reference_dependencies
 - GDS-Utilities, 123
- scan_library_references
 - GDS-Utilities, 124
- search_entry
 - tree_stores, 184
- second
 - gds_time_field, 178
- sel_layer_element_drag_begin
 - LayerSelector Object, 90
- sel_layer_element_drag_data_get
 - LayerSelector Object, 90
- sel_layer_element_drag_end
 - LayerSelector Object, 91
- sel_layer_element_setup_dnd_callbacks
 - LayerSelector Object, 91
- setup_cell_selector
 - Graphical User Interface, 60
- shape_drawer_drawing_callback
 - RendererSettingsDialog, 140
- shape_drawing
 - _RendererSettingsDialog, 161
- show_tex_options
 - RendererSettingsDialog, 140
- SIGNAL_COUNT
 - Graphical User Interface, 51
- SIGNAL_WINDOW_CLOSED
 - Graphical User Interface, 51
- SNAME
 - GDS-Utilities, 105
- sort_down_callback
 - Graphical User Interface, 61
- sort_up_callback
 - Graphical User Interface, 62
- SREF
 - GDS-Utilities, 105
- stacked_position
 - layer_info, 182
- standalone_check
 - _RendererSettingsDialog, 161

- start_gui
 - main.c, [291](#)
- STRANS
 - GDS-Utilities, [105](#)
- STRNAME
 - GDS-Utilities, [105](#)
- super
 - _LibCellRenderer, [159](#)
- tex_pdf_layers
 - render_settings, [183](#)
- tex_standalone
 - render_settings, [183](#)
- tree-store.c, [300](#), [301](#)
- tree-store.h, [303](#), [304](#)
- tree_sel_func
 - Graphical User Interface, [62](#)
- tree_stores, [184](#)
 - base_store, [184](#)
 - base_tree_view, [184](#)
 - filter, [184](#)
 - search_entry, [184](#)
- TYPE_LAYER_ELEMENT
 - LayerElement, [143](#)
- TYPE_LAYER_SELECTOR
 - LayerSelector Object, [72](#)
- TYPE_LIB_CELL_RENDERER
 - LibCellRenderer GObject, [94](#)
- unit_in_meters
 - _RendererSettingsDialog, [161](#)
 - gds_library, [176](#)
- UNITS
 - GDS-Utilities, [105](#)
- unresolved_child_count
 - gds_cell_checks, [170](#)
- update_box_with_gfx
 - Geometric Helper Functions, [42](#)
- upper_right
 - bounding_box::_vectors, [163](#)
- usage.dox, [305](#)
- vector-operations.c, [305](#), [306](#)
- vector-operations.h, [308](#), [309](#)
- vector_2d, [185](#)
 - x, [185](#)
 - y, [185](#)
- vector_2d_abs
 - Geometric Helper Functions, [43](#)
- vector_2d_add
 - Geometric Helper Functions, [43](#)
- vector_2d_alloc
 - Geometric Helper Functions, [44](#)
- vector_2d_calculate_angle_between
 - Geometric Helper Functions, [44](#)
- vector_2d_copy
 - Geometric Helper Functions, [45](#)
- vector_2d_free
 - Geometric Helper Functions, [46](#)
- vector_2d_normalize
 - Geometric Helper Functions, [46](#)
- vector_2d_rotate
 - Geometric Helper Functions, [46](#)
- vector_2d_scalar_multiply
 - Geometric Helper Functions, [47](#)
- vector_2d_scale
 - Geometric Helper Functions, [47](#)
- vector_2d_subtract
 - Geometric Helper Functions, [48](#)
- vector_array
 - bounding_box, [165](#)
- vectors
 - bounding_box, [165](#)
- Version Number, [129](#)
 - _app_version_string, [129](#)
- version.c, [310](#)
- version.h, [310](#), [311](#)
- versioning.dox, [311](#)
- vertices
 - gds_graphics, [174](#)
- widgets.dox, [311](#)
- WIDTH
 - GDS-Utilities, [105](#)
- width_absolute
 - gds_graphics, [174](#)
- write_layer_definitions
 - LaTeX/TikZ Renderer, [67](#)
- write_layer_env
 - LaTeX/TikZ Renderer, [69](#)
- WRITEOUT_BUFFER
 - LaTeX/TikZ Renderer, [64](#)
- x
 - gds_point, [177](#)
 - vector_2d, [185](#)
- x_label
 - _RendererSettingsDialog, [161](#)
- x_output_label
 - _RendererSettingsDialog, [162](#)
- XY
 - GDS-Utilities, [105](#)
- y
 - gds_point, [177](#)
 - vector_2d, [185](#)
- y_label
 - _RendererSettingsDialog, [162](#)
- y_output_label
 - _RendererSettingsDialog, [162](#)
- year
 - gds_time_field, [179](#)