

cairomm

1.18.0

Generated by Doxygen 1.11.0



<b>1 Cairomm: A C++ wrapper for the cairo graphics library</b>	<b>1</b>
1.1 License	1
1.2 Introduction	1
<b>2 New API in cairomm 1.18</b>	<b>3</b>
<b>3 Deprecated List</b>	<b>5</b>
<b>4 Namespace Index</b>	<b>7</b>
4.1 Namespace List	7
<b>5 Hierarchical Index</b>	<b>9</b>
5.1 Class Hierarchy	9
<b>6 Class Index</b>	<b>11</b>
6.1 Class List	11
<b>7 Namespace Documentation</b>	<b>13</b>
7.1 Cairo Namespace Reference	13
7.1.1 Typedef Documentation	16
7.1.1.1 FontExtents	16
7.1.1.2 Glyph	16
7.1.1.3 Rectangle	16
7.1.1.4 RectangleInt	16
7.1.1.5 RefPtr	16
7.1.1.6 TextCluster	17
7.1.1.7 TextExtents	17
7.1.2 Enumeration Type Documentation	17
7.1.2.1 Antialias	17
7.1.2.2 Content	17
7.1.2.3 FontType	17
7.1.2.4 FtSynthesize	18
7.1.2.5 PdfVersion	18
7.1.2.6 PsLevel	18
7.1.2.7 SubpixelOrder	19
7.1.2.8 SvgVersion	19
7.1.2.9 TextClusterFlags	19
7.1.3 Function Documentation	20
7.1.3.1 make_refptr_for_instance()	20
7.1.3.2 operator&()	20
7.1.3.3 operator"   ()	20
<b>8 Class Documentation</b>	<b>21</b>
8.1 Cairo::ColorStop Struct Reference	21
8.1.1 Member Data Documentation	21

8.1.1.1 alpha	21
8.1.1.2 blue	21
8.1.1.3 green	21
8.1.1.4 offset	21
8.1.1.5 red	22
8.2 Cairo::Context Class Reference	22
8.2.1 Detailed Description	28
8.2.2 Member Typedef Documentation	28
8.2.2.1 cobject	28
8.2.3 Member Enumeration Documentation	28
8.2.3.1 FillRule	28
8.2.3.2 LineCap	29
8.2.3.3 LineJoin	29
8.2.3.4 Operator	29
8.2.4 Constructor & Destructor Documentation	30
8.2.4.1 Context() [1/3]	30
8.2.4.2 Context() [2/3]	30
8.2.4.3 Context() [3/3]	30
8.2.4.4 ~Context()	31
8.2.5 Member Function Documentation	31
8.2.5.1 append_path()	31
8.2.5.2 arc()	31
8.2.5.3 arc_negative()	32
8.2.5.4 begin_new_path()	32
8.2.5.5 begin_new_sub_path()	32
8.2.5.6 clip()	33
8.2.5.7 clip_preserve()	33
8.2.5.8 close_path()	33
8.2.5.9 cobj() [1/2]	34
8.2.5.10 cobj() [2/2]	34
8.2.5.11 copy_clip_rectangle_list()	34
8.2.5.12 copy_page()	34
8.2.5.13 copy_path()	35
8.2.5.14 copy_path_flat()	35
8.2.5.15 create()	35
8.2.5.16 curve_to()	35
8.2.5.17 device_to_user()	36
8.2.5.18 device_to_user_distance()	36
8.2.5.19 fill()	36
8.2.5.20 fill_preserve()	37
8.2.5.21 get_antialias()	37
8.2.5.22 get_clip_extents()	37

8.2.5.23	<a href="#">get_current_point()</a>	37
8.2.5.24	<a href="#">get_dash()</a>	38
8.2.5.25	<a href="#">get_fill_extents()</a>	38
8.2.5.26	<a href="#">get_fill_rule()</a>	39
8.2.5.27	<a href="#">get_font_extents()</a>	39
8.2.5.28	<a href="#">get_font_face()</a> [1/2]	39
8.2.5.29	<a href="#">get_font_face()</a> [2/2]	39
8.2.5.30	<a href="#">get_font_matrix()</a>	39
8.2.5.31	<a href="#">get_font_options()</a>	39
8.2.5.32	<a href="#">get_glyph_extents()</a>	40
8.2.5.33	<a href="#">get_group_target()</a> [1/2]	40
8.2.5.34	<a href="#">get_group_target()</a> [2/2]	41
8.2.5.35	<a href="#">get_line_cap()</a>	41
8.2.5.36	<a href="#">get_line_join()</a>	41
8.2.5.37	<a href="#">get_line_width()</a>	41
8.2.5.38	<a href="#">get_matrix()</a> [1/2]	41
8.2.5.39	<a href="#">get_matrix()</a> [2/2]	41
8.2.5.40	<a href="#">get_miter_limit()</a>	42
8.2.5.41	<a href="#">get_operator()</a>	42
8.2.5.42	<a href="#">get_path_extents()</a>	42
8.2.5.43	<a href="#">get_scaled_font()</a>	43
8.2.5.44	<a href="#">get_source()</a> [1/2]	43
8.2.5.45	<a href="#">get_source()</a> [2/2]	43
8.2.5.46	<a href="#">get_source_for_surface()</a> [1/2]	43
8.2.5.47	<a href="#">get_source_for_surface()</a> [2/2]	43
8.2.5.48	<a href="#">get_stroke_extents()</a>	43
8.2.5.49	<a href="#">get_target()</a> [1/2]	44
8.2.5.50	<a href="#">get_target()</a> [2/2]	44
8.2.5.51	<a href="#">get_text_extents()</a>	44
8.2.5.52	<a href="#">get_tolerance()</a>	45
8.2.5.53	<a href="#">glyph_path()</a>	45
8.2.5.54	<a href="#">has_current_point()</a>	45
8.2.5.55	<a href="#">in_clip()</a>	45
8.2.5.56	<a href="#">in_fill()</a>	46
8.2.5.57	<a href="#">in_stroke()</a>	46
8.2.5.58	<a href="#">line_to()</a>	47
8.2.5.59	<a href="#">mask()</a> [1/2]	47
8.2.5.60	<a href="#">mask()</a> [2/2]	47
8.2.5.61	<a href="#">move_to()</a>	48
8.2.5.62	<a href="#">operator=()</a>	48
8.2.5.63	<a href="#">paint()</a>	48
8.2.5.64	<a href="#">paint_with_alpha()</a>	48

8.2.5.65 pop_group()	49
8.2.5.66 pop_group_to_source()	49
8.2.5.67 push_group()	49
8.2.5.68 push_group_with_content()	50
8.2.5.69 rectangle()	50
8.2.5.70 rel_curve_to()	51
8.2.5.71 rel_line_to()	51
8.2.5.72 rel_move_to()	52
8.2.5.73 reset_clip()	52
8.2.5.74 restore()	52
8.2.5.75 rotate()	52
8.2.5.76 rotate_degrees()	53
8.2.5.77 save()	53
8.2.5.78 scale()	53
8.2.5.79 select_font_face()	54
8.2.5.80 set_antialias()	54
8.2.5.81 set_dash() [1/2]	55
8.2.5.82 set_dash() [2/2]	55
8.2.5.83 set_fill_rule()	55
8.2.5.84 set_font_face()	56
8.2.5.85 set_font_matrix()	56
8.2.5.86 set_font_options()	56
8.2.5.87 set_font_size()	57
8.2.5.88 set_identity_matrix()	58
8.2.5.89 set_line_cap()	58
8.2.5.90 set_line_join()	58
8.2.5.91 set_line_width()	59
8.2.5.92 set_matrix()	59
8.2.5.93 set_miter_limit()	59
8.2.5.94 set_operator()	60
8.2.5.95 set_scaled_font()	60
8.2.5.96 set_source() [1/2]	60
8.2.5.97 set_source() [2/2]	61
8.2.5.98 set_source_rgb()	61
8.2.5.99 set_source_rgba()	62
8.2.5.100 set_tolerance()	62
8.2.5.101 show_glyphs()	63
8.2.5.102 show_page()	63
8.2.5.103 show_text()	63
8.2.5.104 show_text_glyphs()	64
8.2.5.105 stroke()	64
8.2.5.106 stroke_preserve()	65

8.2.5.107 text_path()	65
8.2.5.108 transform()	66
8.2.5.109 translate()	66
8.2.5.110 unset_dash()	66
8.2.5.111 user_to_device()	66
8.2.5.112 user_to_device_distance()	67
8.2.6 Member Data Documentation	67
8.2.6.1 m_cobject	67
8.3 Cairo::Device Class Reference	67
8.3.1 Detailed Description	68
8.3.2 Member Typedef Documentation	69
8.3.2.1 cobject	69
8.3.3 Member Enumeration Documentation	69
8.3.3.1 DeviceType	69
8.3.4 Constructor & Destructor Documentation	69
8.3.4.1 Device()	69
8.3.4.2 ~Device()	69
8.3.5 Member Function Documentation	70
8.3.5.1 acquire()	70
8.3.5.2 cobj() [1/2]	70
8.3.5.3 cobj() [2/2]	70
8.3.5.4 finish()	70
8.3.5.5 flush()	71
8.3.5.6 get_type()	71
8.3.5.7 reference()	71
8.3.5.8 release()	71
8.3.5.9 unreference()	71
8.3.6 Member Data Documentation	71
8.3.6.1 m_cobject	71
8.4 Cairo::Device::Lock Class Reference	71
8.4.1 Detailed Description	72
8.4.2 Constructor & Destructor Documentation	72
8.4.2.1 Lock() [1/2]	72
8.4.2.2 Lock() [2/2]	72
8.4.2.3 ~Lock()	72
8.5 Cairo::FontFace Class Reference	73
8.5.1 Detailed Description	74
8.5.2 Member Typedef Documentation	74
8.5.2.1 cobject	74
8.5.3 Constructor & Destructor Documentation	74
8.5.3.1 FontFace() [1/2]	74
8.5.3.2 FontFace() [2/2]	74

8.5.3.3 ~FontFace()	74
8.5.4 Member Function Documentation	74
8.5.4.1 cobj() [1/2]	74
8.5.4.2 cobj() [2/2]	75
8.5.4.3 get_type()	75
8.5.4.4 operator=()	75
8.5.4.5 reference()	75
8.5.4.6 unreference()	75
8.5.5 Member Data Documentation	75
8.5.5.1 m_cobject	75
8.6 Cairo::FontOptions Class Reference	75
8.6.1 Detailed Description	77
8.6.2 Member Typedef Documentation	77
8.6.2.1 cobject	77
8.6.3 Member Enumeration Documentation	77
8.6.3.1 HintMetrics	77
8.6.3.2 HintStyle	77
8.6.4 Constructor & Destructor Documentation	78
8.6.4.1 FontOptions() [1/3]	78
8.6.4.2 FontOptions() [2/3]	78
8.6.4.3 FontOptions() [3/3]	78
8.6.4.4 ~FontOptions()	78
8.6.5 Member Function Documentation	78
8.6.5.1 cobj() [1/2]	78
8.6.5.2 cobj() [2/2]	78
8.6.5.3 get_antialias()	78
8.6.5.4 get_hint_metrics()	79
8.6.5.5 get_hint_style()	79
8.6.5.6 get_subpixel_order()	79
8.6.5.7 hash()	79
8.6.5.8 merge()	79
8.6.5.9 operator=()	80
8.6.5.10 operator==()	80
8.6.5.11 set_antialias()	80
8.6.5.12 set_hint_metrics()	80
8.6.5.13 set_hint_style()	80
8.6.5.14 set_subpixel_order()	81
8.6.6 Member Data Documentation	81
8.6.6.1 m_cobject	81
8.7 Cairo::FtFontFace Class Reference	81
8.7.1 Constructor & Destructor Documentation	82
8.7.1.1 FtFontFace()	82



8.7.2 Member Function Documentation	83
8.7.2.1 create()	83
8.7.2.2 get_synthesize()	83
8.7.2.3 set_synthesize()	83
8.7.2.4 unset_synthesize()	84
8.8 Cairo::FtScaledFont Class Reference	84
8.8.1 Detailed Description	86
8.8.2 Constructor & Destructor Documentation	86
8.8.2.1 FtScaledFont()	86
8.8.3 Member Function Documentation	86
8.8.3.1 create()	86
8.8.3.2 lock_face()	87
8.8.3.3 unlock_face()	87
8.9 Cairo::GlitzSurface Class Reference	87
8.9.1 Detailed Description	90
8.9.2 Constructor & Destructor Documentation	91
8.9.2.1 GlitzSurface()	91
8.9.2.2 ~GlitzSurface()	91
8.9.3 Member Function Documentation	91
8.9.3.1 create()	91
8.10 Cairo::Gradient Class Reference	92
8.10.1 Constructor & Destructor Documentation	93
8.10.1.1 Gradient() [1/2]	93
8.10.1.2 ~Gradient()	94
8.10.1.3 Gradient() [2/2]	94
8.10.2 Member Function Documentation	94
8.10.2.1 add_color_stop_rgb()	94
8.10.2.2 add_color_stop_rgba()	95
8.10.2.3 get_color_stops()	95
8.11 Cairo::ImageSurface Class Reference	96
8.11.1 Detailed Description	99
8.11.2 Constructor & Destructor Documentation	100
8.11.2.1 ImageSurface()	100
8.11.2.2 ~ImageSurface()	100
8.11.3 Member Function Documentation	100
8.11.3.1 create() [1/2]	100
8.11.3.2 create() [2/2]	101
8.11.3.3 create_from_png()	101
8.11.3.4 create_from_png_stream()	102
8.11.3.5 format_stride_for_width()	102
8.11.3.6 get_data() [1/2]	103
8.11.3.7 get_data() [2/2]	103

8.11.3.8 <code>get_format()</code>	103
8.11.3.9 <code>get_height()</code>	103
8.11.3.10 <code>get_stride()</code>	103
8.11.3.11 <code>get_width()</code>	104
8.12 Cairo::LinearGradient Class Reference	104
8.12.1 Constructor & Destructor Documentation	106
8.12.1.1 <code>LinearGradient()</code> [1/2]	106
8.12.1.2 <code>LinearGradient()</code> [2/2]	106
8.12.1.3 <code>~LinearGradient()</code>	107
8.12.2 Member Function Documentation	107
8.12.2.1 <code>create()</code>	107
8.12.2.2 <code>get_linear_points()</code>	107
8.13 Cairo::logic_error Class Reference	108
8.13.1 Constructor & Destructor Documentation	108
8.13.1.1 <code>logic_error()</code>	108
8.13.1.2 <code>~logic_error()</code>	108
8.13.2 Member Function Documentation	109
8.13.2.1 <code>get_status_code()</code>	109
8.14 Cairo::Matrix Class Reference	109
8.14.1 Detailed Description	110
8.14.2 Constructor & Destructor Documentation	110
8.14.2.1 <code>Matrix()</code> [1/2]	110
8.14.2.2 <code>Matrix()</code> [2/2]	111
8.14.3 Member Function Documentation	111
8.14.3.1 <code>invert()</code>	111
8.14.3.2 <code>multiply()</code>	111
8.14.3.3 <code>rotate()</code>	112
8.14.3.4 <code>scale()</code>	112
8.14.3.5 <code>transform_distance()</code>	112
8.14.3.6 <code>transform_point()</code>	113
8.14.3.7 <code>translate()</code>	113
8.14.4 Friends And Related Symbol Documentation	113
8.14.4.1 <code>identity_matrix()</code>	113
8.14.4.2 <code>operator*()</code>	113
8.14.4.3 <code>rotation_matrix()</code>	114
8.14.4.4 <code>scaling_matrix()</code>	114
8.14.4.5 <code>translation_matrix()</code>	114
8.15 Cairo::Path Class Reference	115
8.15.1 Detailed Description	115
8.15.2 Member Typedef Documentation	115
8.15.2.1 <code>cobject</code>	115
8.15.3 Constructor & Destructor Documentation	115

8.15.3.1 Path() [1/2]	115
8.15.3.2 Path() [2/2]	116
8.15.3.3 ~Path()	116
8.15.4 Member Function Documentation	116
8.15.4.1 cobj() [1/2]	116
8.15.4.2 cobj() [2/2]	116
8.15.4.3 operator=()	116
8.15.5 Member Data Documentation	116
8.15.5.1 m_cobject	116
8.16 Cairo::Pattern Class Reference	116
8.16.1 Detailed Description	118
8.16.2 Member Typedef Documentation	118
8.16.2.1 cobject	118
8.16.3 Member Enumeration Documentation	118
8.16.3.1 Extend	118
8.16.3.2 Type	118
8.16.4 Constructor & Destructor Documentation	119
8.16.4.1 Pattern() [1/3]	119
8.16.4.2 Pattern() [2/3]	119
8.16.4.3 ~Pattern()	119
8.16.4.4 Pattern() [3/3]	119
8.16.5 Member Function Documentation	119
8.16.5.1 cobj() [1/2]	119
8.16.5.2 cobj() [2/2]	120
8.16.5.3 get_extend()	120
8.16.5.4 get_matrix() [1/2]	120
8.16.5.5 get_matrix() [2/2]	120
8.16.5.6 get_type()	120
8.16.5.7 operator=()	120
8.16.5.8 reference()	121
8.16.5.9 set_extend()	121
8.16.5.10 set_matrix()	122
8.16.5.11 unreference()	122
8.16.6 Member Data Documentation	122
8.16.6.1 m_cobject	122
8.17 Cairo::PdfSurface Class Reference	123
8.17.1 Detailed Description	126
8.17.2 Constructor & Destructor Documentation	126
8.17.2.1 PdfSurface()	126
8.17.2.2 ~PdfSurface()	127
8.17.3 Member Function Documentation	127
8.17.3.1 create()	127

8.17.3.2 create_for_stream()	127
8.17.3.3 get_versions()	128
8.17.3.4 restrict_to_version()	128
8.17.3.5 set_size()	128
8.17.3.6 version_to_string()	129
8.18 Cairo::PsSurface Class Reference	129
8.18.1 Detailed Description	133
8.18.2 Constructor & Destructor Documentation	133
8.18.2.1 PsSurface()	133
8.18.2.2 ~PsSurface()	133
8.18.3 Member Function Documentation	133
8.18.3.1 create()	133
8.18.3.2 create_for_stream()	134
8.18.3.3 dsc_begin_page_setup()	134
8.18.3.4 dsc_begin_setup()	134
8.18.3.5 dsc_comment()	134
8.18.3.6 get_eps()	135
8.18.3.7 get_levels()	135
8.18.3.8 level_to_string()	135
8.18.3.9 restrict_to_level()	135
8.18.3.10 set_eps()	136
8.18.3.11 set_size()	136
8.19 Cairo::QuartzFontFace Class Reference	137
8.19.1 Detailed Description	138
8.19.2 Constructor & Destructor Documentation	138
8.19.2.1 QuartzFontFace() [1/2]	138
8.19.2.2 QuartzFontFace() [2/2]	138
8.19.3 Member Function Documentation	138
8.19.3.1 create() [1/2]	138
8.19.3.2 create() [2/2]	138
8.20 Cairo::QuartzSurface Class Reference	139
8.20.1 Detailed Description	142
8.20.2 Constructor & Destructor Documentation	142
8.20.2.1 QuartzSurface()	142
8.20.2.2 ~QuartzSurface()	143
8.20.3 Member Function Documentation	143
8.20.3.1 create() [1/2]	143
8.20.3.2 create() [2/2]	143
8.20.3.3 get_cg_context()	144
8.21 Cairo::RadialGradient Class Reference	144
8.21.1 Constructor & Destructor Documentation	146
8.21.1.1 RadialGradient() [1/2]	146

8.21.1.2 RadialGradient() [2/2]	146
8.21.1.3 ~RadialGradient()	147
8.21.2 Member Function Documentation	147
8.21.2.1 create()	147
8.21.2.2 get_radial_circles()	147
8.22 Cairo::RecordingSurface Class Reference	148
8.22.1 Detailed Description	151
8.22.2 Constructor & Destructor Documentation	152
8.22.2.1 RecordingSurface()	152
8.22.2.2 ~RecordingSurface()	152
8.22.3 Member Function Documentation	152
8.22.3.1 create() [1/2]	152
8.22.3.2 create() [2/2]	152
8.22.3.3 get_extents()	153
8.22.3.4 ink_extents()	153
8.23 Cairo::Region Class Reference	153
8.23.1 Detailed Description	155
8.23.2 Member Typedef Documentation	155
8.23.2.1 cobject	155
8.23.3 Member Enumeration Documentation	155
8.23.3.1 Overlap	155
8.23.4 Constructor & Destructor Documentation	155
8.23.4.1 Region()	155
8.23.4.2 ~Region()	156
8.23.5 Member Function Documentation	156
8.23.5.1 cobj() [1/2]	156
8.23.5.2 cobj() [2/2]	156
8.23.5.3 contains_point()	156
8.23.5.4 contains_rectangle()	156
8.23.5.5 copy()	156
8.23.5.6 create() [1/4]	156
8.23.5.7 create() [2/4]	157
8.23.5.8 create() [3/4]	157
8.23.5.9 create() [4/4]	157
8.23.5.10 do_union() [1/2]	157
8.23.5.11 do_union() [2/2]	157
8.23.5.12 do_xor() [1/2]	157
8.23.5.13 do_xor() [2/2]	158
8.23.5.14 empty()	158
8.23.5.15 get_extents()	158
8.23.5.16 get_num_rectangles()	158
8.23.5.17 get_rectangle()	158

8.23.5.18 intersect() [1/2]	158
8.23.5.19 intersect() [2/2]	158
8.23.5.20 reference()	159
8.23.5.21 subtract() [1/2]	159
8.23.5.22 subtract() [2/2]	159
8.23.5.23 translate()	159
8.23.5.24 unreference()	159
8.23.6 Member Data Documentation	159
8.23.6.1 m_cobject	159
8.24 Cairo::SaveGuard Class Reference	159
8.24.1 Detailed Description	160
8.24.2 Constructor & Destructor Documentation	160
8.24.2.1 SaveGuard()	160
8.24.2.2 ~SaveGuard()	160
8.25 Cairo::ScaledFont Class Reference	161
8.25.1 Detailed Description	162
8.25.2 Member Typedef Documentation	162
8.25.2.1 cobject	162
8.25.3 Constructor & Destructor Documentation	162
8.25.3.1 ScaledFont() [1/3]	162
8.25.3.2 ScaledFont() [2/3]	163
8.25.3.3 ~ScaledFont()	163
8.25.3.4 ScaledFont() [3/3]	163
8.25.4 Member Function Documentation	163
8.25.4.1 cobj() [1/2]	163
8.25.4.2 cobj() [2/2]	163
8.25.4.3 create()	163
8.25.4.4 get_ctm()	164
8.25.4.5 get_extents()	164
8.25.4.6 get_font_face()	164
8.25.4.7 get_font_matrix()	164
8.25.4.8 get_font_options()	165
8.25.4.9 get_glyph_extents()	165
8.25.4.10 get_scale_matrix()	165
8.25.4.11 get_text_extents()	166
8.25.4.12 get_type()	166
8.25.4.13 operator=()	166
8.25.4.14 text_to_glyphs()	166
8.25.5 Member Data Documentation	167
8.25.5.1 m_cobject	167
8.26 Cairo::SolidPattern Class Reference	167
8.26.1 Constructor & Destructor Documentation	169

8.26.1.1 SolidPattern()	169
8.26.1.2 ~SolidPattern()	169
8.26.2 Member Function Documentation	169
8.26.2.1 create_rgb()	169
8.26.2.2 create_rgba()	170
8.26.2.3 get_rgba()	170
8.27 Cairo::Surface Class Reference	171
8.27.1 Detailed Description	174
8.27.2 Member Typedef Documentation	174
8.27.2.1 cobject	174
8.27.2.2 SlotDestroy	174
8.27.2.3 SlotReadFunc	174
8.27.2.4 SlotWriteFunc	175
8.27.3 Member Enumeration Documentation	175
8.27.3.1 Format	175
8.27.3.2 Type	176
8.27.4 Constructor & Destructor Documentation	177
8.27.4.1 Surface() [1/2]	177
8.27.4.2 Surface() [2/2]	178
8.27.4.3 ~Surface()	178
8.27.5 Member Function Documentation	178
8.27.5.1 cobj() [1/2]	178
8.27.5.2 cobj() [2/2]	178
8.27.5.3 copy_page()	178
8.27.5.4 create() [1/2]	179
8.27.5.5 create() [2/2]	180
8.27.5.6 finish()	180
8.27.5.7 flush()	181
8.27.5.8 get_content()	181
8.27.5.9 get_device()	181
8.27.5.10 get_device_offset()	181
8.27.5.11 get_device_scale() [1/2]	181
8.27.5.12 get_device_scale() [2/2]	182
8.27.5.13 get_fallback_resolution()	182
8.27.5.14 get_font_options()	182
8.27.5.15 get_mime_data()	182
8.27.5.16 get_type()	183
8.27.5.17 has_show_text_glyphs()	183
8.27.5.18 mark_dirty() [1/2]	183
8.27.5.19 mark_dirty() [2/2]	183
8.27.5.20 operator=()	184
8.27.5.21 set_device_offset()	184

8.27.5.22 set_device_scale() [1/2]	184
8.27.5.23 set_device_scale() [2/2]	185
8.27.5.24 set_fallback_resolution()	185
8.27.5.25 set_mime_data()	186
8.27.5.26 show_page()	186
8.27.5.27 unset_mime_data()	186
8.27.5.28 write_to_png()	187
8.27.5.29 write_to_png_stream()	187
8.27.6 Member Data Documentation	187
8.27.6.1 m_cobject	187
8.28 Cairo::SurfacePattern Class Reference	188
8.28.1 Member Enumeration Documentation	190
8.28.1.1 Filter	190
8.28.2 Constructor & Destructor Documentation	190
8.28.2.1 SurfacePattern() [1/2]	190
8.28.2.2 SurfacePattern() [2/2]	190
8.28.2.3 ~SurfacePattern()	191
8.28.3 Member Function Documentation	191
8.28.3.1 create()	191
8.28.3.2 get_filter()	191
8.28.3.3 get_surface() [1/2]	191
8.28.3.4 get_surface() [2/2]	191
8.28.3.5 set_filter()	191
8.29 Cairo::SvgSurface Class Reference	192
8.29.1 Detailed Description	195
8.29.2 Constructor & Destructor Documentation	195
8.29.2.1 SvgSurface()	195
8.29.2.2 ~SvgSurface()	196
8.29.3 Member Function Documentation	196
8.29.3.1 create()	196
8.29.3.2 create_for_stream()	196
8.29.3.3 get_versions()	197
8.29.3.4 restrict_to_version()	197
8.29.3.5 version_to_string()	197
8.30 Cairo::ToyFontFace Class Reference	198
8.30.1 Detailed Description	199
8.30.2 Member Enumeration Documentation	199
8.30.2.1 Slant	199
8.30.2.2 Weight	200
8.30.3 Constructor & Destructor Documentation	200
8.30.3.1 ToyFontFace()	200
8.30.4 Member Function Documentation	200



8.30.4.1 create()	200
8.30.4.2 get_family()	201
8.30.4.3 get_slant()	201
8.30.4.4 get_weight()	201
8.31 Cairo::UserFontFace Class Reference	201
8.31.1 Detailed Description	203
8.31.2 Constructor & Destructor Documentation	203
8.31.2.1 ~UserFontFace()	203
8.31.2.2 UserFontFace()	204
8.31.3 Member Function Documentation	204
8.31.3.1 init()	204
8.31.3.2 render_glyph()	205
8.31.3.3 text_to_glyphs()	206
8.31.3.4 unicode_to_glyph()	207
8.32 Cairo::Win32FontFace Class Reference	208
8.32.1 Detailed Description	209
8.32.2 Constructor & Destructor Documentation	209
8.32.2.1 Win32FontFace() [1/3]	209
8.32.2.2 Win32FontFace() [2/3]	209
8.32.2.3 Win32FontFace() [3/3]	209
8.32.3 Member Function Documentation	209
8.32.3.1 create() [1/3]	209
8.32.3.2 create() [2/3]	210
8.32.3.3 create() [3/3]	210
8.33 Cairo::Win32PrintingSurface Class Reference	211
8.33.1 Detailed Description	214
8.33.2 Constructor & Destructor Documentation	214
8.33.2.1 Win32PrintingSurface()	214
8.33.2.2 ~Win32PrintingSurface()	214
8.33.3 Member Function Documentation	214
8.33.3.1 create()	214
8.34 Cairo::Win32ScaledFont Class Reference	215
8.34.1 Detailed Description	217
8.34.2 Constructor & Destructor Documentation	217
8.34.2.1 Win32ScaledFont()	217
8.34.3 Member Function Documentation	217
8.34.3.1 create()	217
8.34.3.2 done_font()	217
8.34.3.3 get_device_to_logical()	217
8.34.3.4 get_logical_to_device()	218
8.34.3.5 get_metrics_factor()	218
8.34.3.6 select_font()	218

8.35 Cairo::Win32Surface Class Reference . . . . .	219
8.35.1 Detailed Description . . . . .	222
8.35.2 Constructor & Destructor Documentation . . . . .	222
8.35.2.1 Win32Surface() . . . . .	222
8.35.2.2 ~Win32Surface() . . . . .	223
8.35.3 Member Function Documentation . . . . .	223
8.35.3.1 create() . . . . .	223
8.35.3.2 create_with_ddb() . . . . .	223
8.35.3.3 create_with_dib() . . . . .	224
8.35.3.4 get_dc() . . . . .	224
8.35.3.5 get_image() . . . . .	224
8.36 Cairo::XlibSurface Class Reference . . . . .	225
8.36.1 Detailed Description . . . . .	228
8.36.2 Constructor & Destructor Documentation . . . . .	228
8.36.2.1 XlibSurface() . . . . .	228
8.36.2.2 ~XlibSurface() . . . . .	229
8.36.3 Member Function Documentation . . . . .	229
8.36.3.1 create() [1/2] . . . . .	229
8.36.3.2 create() [2/2] . . . . .	229
8.36.3.3 create_with_xrender_format() . . . . .	230
8.36.3.4 get_depth() . . . . .	230
8.36.3.5 get_display() [1/2] . . . . .	231
8.36.3.6 get_display() [2/2] . . . . .	231
8.36.3.7 get_drawable() . . . . .	231
8.36.3.8 get_height() . . . . .	231
8.36.3.9 get_screen() [1/2] . . . . .	231
8.36.3.10 get_screen() [2/2] . . . . .	231
8.36.3.11 get_visual() [1/2] . . . . .	231
8.36.3.12 get_visual() [2/2] . . . . .	231
8.36.3.13 get_width() . . . . .	232
8.36.3.14 get_xrender_format() . . . . .	232
8.36.3.15 set_drawable() . . . . .	232
8.36.3.16 set_size() . . . . .	232
8.37 cairo_matrix_t Class Reference . . . . .	233
8.37.1 Detailed Description . . . . .	233
8.38 hash_load_check_resize_trigger_size_base Class Reference . . . . .	233
8.39 lu_counter_policy_base Class Reference . . . . .	234
8.40 mask_based_range_hashing Class Reference . . . . .	234
8.41 mod_based_range_hashing Class Reference . . . . .	235
<b>9 Examples . . . . .</b>	<b>237</b>
9.1 toy-text.cc . . . . .	237

---

9.2 user-font.cc . . . . .	237
9.3 image-surface.cc . . . . .	239
9.4 pdf-surface.cc . . . . .	240
9.5 ps-surface.cc . . . . .	241
9.6 svg-surface.cc . . . . .	242
<b>Index</b>	<b>243</b>



## Chapter 1

# Cairomm: A C++ wrapper for the cairo graphics library

### 1.1 License

Cairomm is available under the terms of the LGPL license

### 1.2 Introduction

If you're just beginning to learn cairomm, a good place to start is with the [Cairo::Surface](#) and [Cairo::Context](#) classes. In general terms, you draw onto a Surface using the graphics settings specified in your Context.



## Chapter 2

# New API in cairomm 1.18

Class `Cairo::SaveGuard`

Member `Cairo::Surface::get_device_scale` (double &x\_scale, double &y\_scale) const

Member `Cairo::Surface::get_device_scale` () const

Member `Cairo::Surface::set_device_scale` (double x\_scale, double y\_scale)

Member `Cairo::Surface::set_device_scale` (double scale)





## Chapter 3

# Deprecated List

Member [Cairo::Surface::WIN32](#)

Use WIN32\_SURFACE instead.



## Chapter 4

# Namespace Index

### 4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">Cairo</a> . . . . .	13
---------------------------------	----



## Chapter 5

# Hierarchical Index

### 5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Cairo::ColorStop . . . . .	21
Cairo::Context . . . . .	22
Cairo::Device . . . . .	67
Cairo::Device::Lock . . . . .	71
Cairo::FontFace . . . . .	73
Cairo::FtFontFace . . . . .	81
Cairo::QuartzFontFace . . . . .	137
Cairo::ToyFontFace . . . . .	198
Cairo::UserFontFace . . . . .	201
Cairo::Win32FontFace . . . . .	208
Cairo::FontOptions . . . . .	75
Cairo::Path . . . . .	115
Cairo::Pattern . . . . .	116
Cairo::Gradient . . . . .	92
Cairo::LinearGradient . . . . .	104
Cairo::RadialGradient . . . . .	144
Cairo::SolidPattern . . . . .	167
Cairo::SurfacePattern . . . . .	188
Cairo::Region . . . . .	153
Cairo::SaveGuard . . . . .	159
Cairo::ScaledFont . . . . .	161
Cairo::FtScaledFont . . . . .	84
Cairo::Win32ScaledFont . . . . .	215
Cairo::Surface . . . . .	171
Cairo::GlitzSurface . . . . .	87
Cairo::ImageSurface . . . . .	96
Cairo::PdfSurface . . . . .	123
Cairo::PsSurface . . . . .	129
Cairo::QuartzSurface . . . . .	139
Cairo::RecordingSurface . . . . .	148
Cairo::SvgSurface . . . . .	192
Cairo::Win32PrintingSurface . . . . .	211
Cairo::Win32Surface . . . . .	219
Cairo::XlibSurface . . . . .	225

cairo_matrix_t . . . . .	233
Cairo::Matrix . . . . .	109
hash_load_check_resize_trigger_size_base . . . . .	233
lu_counter_policy_base . . . . .	234
mask_based_range_hashing . . . . .	234
mod_based_range_hashing . . . . .	235
std::exception[external]	
std::logic_error[external]	
Cairo::logic_error . . . . .	108

## Chapter 6

# Class Index

### 6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Cairo::ColorStop</a> . . . . .	21
<a href="#">Cairo::Context</a> Context is the main class used to draw in cairomm . . . . .	22
<a href="#">Cairo::Device</a> Devices are the abstraction <a href="#">Cairo</a> employs for the rendering system used by a <code>cairo_surface_t</code> . . . . .	67
<a href="#">Cairo::Device::Lock</a> A convenience class for acquiring a <a href="#">Device</a> object in an exception-safe manner . . . . .	71
<a href="#">Cairo::FontFace</a> A <a href="#">FontFace</a> represents a particular font at a particular weight, slant, and other characteristic but no size, transformation, or size . . . . .	73
<a href="#">Cairo::FontOptions</a> The font options specify how fonts should be rendered . . . . .	75
<a href="#">Cairo::FtFontFace</a> . . . . .	81
<a href="#">Cairo::FtScaledFont</a> . . . . .	84
<a href="#">Cairo::GlitzSurface</a> A <a href="#">GlitzSurface</a> provides a way to render to the X Window System using Glitz . . . . .	87
<a href="#">Cairo::Gradient</a> . . . . .	92
<a href="#">Cairo::ImageSurface</a> Image surfaces provide the ability to render to memory buffers either allocated by cairo or by the calling code . . . . .	96
<a href="#">Cairo::LinearGradient</a> . . . . .	104
<a href="#">Cairo::logic_error</a> . . . . .	108
<a href="#">Cairo::Matrix</a> A Transformation matrix . . . . .	109
<a href="#">Cairo::Path</a> A data structure for holding a path . . . . .	115
<a href="#">Cairo::Pattern</a> <a href="#">Cairo::Pattern</a> is the paint with which cairo draws . . . . .	116
<a href="#">Cairo::PdfSurface</a> A <a href="#">PdfSurface</a> provides a way to render PDF documents from cairo . . . . .	123
<a href="#">Cairo::PsSurface</a> A <a href="#">PsSurface</a> provides a way to render PostScript documents from cairo . . . . .	129
<a href="#">Cairo::QuartzFontFace</a> The Quartz font backend is primarily used to render text on Apple MacOS X systems . . . . .	137
<a href="#">Cairo::QuartzSurface</a> A <a href="#">QuartzSurface</a> provides a way to render within Apple Mac OS X . . . . .	139

<a href="#">Cairo::RadialGradient</a>	144
<a href="#">Cairo::RecordingSurface</a>	
A recording surface is a surface that records all drawing operations at the highest level of the surface backend interface, (that is, the level of paint, mask, stroke, fill, and show_text_glyphs)	148
<a href="#">Cairo::Region</a>	
A simple graphical data type representing an area of integer-aligned rectangles	153
<a href="#">Cairo::SaveGuard</a>	
RAII-style context save/restore class	159
<a href="#">Cairo::ScaledFont</a>	
A <a href="#">ScaledFont</a> is a font scaled to a particular size and device resolution	161
<a href="#">Cairo::SolidPattern</a>	167
<a href="#">Cairo::Surface</a>	
A cairo surface represents an image, either as the destination of a drawing operation or as source when drawing onto another surface	171
<a href="#">Cairo::SurfacePattern</a>	188
<a href="#">Cairo::SvgSurface</a>	
A <a href="#">SvgSurface</a> provides a way to render Scalable Vector Graphics (SVG) images from cairo	192
<a href="#">Cairo::ToyFontFace</a>	
A simple font face used for the cairo 'toy' font API	198
<a href="#">Cairo::UserFontFace</a>	
Font support with font data provided by the user	201
<a href="#">Cairo::Win32FontFace</a>	
Font support for Microsoft Windows	208
<a href="#">Cairo::Win32PrintingSurface</a>	
A multi-page vector surface type for printing on Microsoft Windows	211
<a href="#">Cairo::Win32ScaledFont</a>	
Scaled Font implementation for Microsoft Windows fonts	215
<a href="#">Cairo::Win32Surface</a>	
A <a href="#">Win32Surface</a> provides a way to render within Microsoft Windows	219
<a href="#">Cairo::XlibSurface</a>	
An <a href="#">XlibSurface</a> provides a way to render to the X Window System using XLib	225
<a href="#">cairo_matrix_t</a>	
See the <a href="#">cairo_matrix_t reference</a> in the cairo manual for more information	233
<a href="#">hash_load_check_resize_trigger_size_base</a>	233
<a href="#">lu_counter_policy_base</a>	234
<a href="#">mask_based_range_hashing</a>	234
<a href="#">mod_based_range_hashing</a>	235



# Chapter 7

## Namespace Documentation

### 7.1 Cairo Namespace Reference

#### Classes

- struct [ColorStop](#)
- class [Context](#)  
*[Context](#) is the main class used to draw in cairomm.*
- class [Device](#)  
*Devices are the abstraction [Cairo](#) employs for the rendering system used by a [cairo\\_surface\\_t](#).*
- class [FontFace](#)  
*A [FontFace](#) represents a particular font at a particular weight, slant, and other characteristic but no size, transformation, or size.*
- class [FontOptions](#)  
*The font options specify how fonts should be rendered.*
- class [FtFontFace](#)
- class [FtScaledFont](#)
- class [GlitzSurface](#)  
*A [GlitzSurface](#) provides a way to render to the X Window System using Glitz.*
- class [Gradient](#)
- class [ImageSurface](#)  
*Image surfaces provide the ability to render to memory buffers either allocated by cairo or by the calling code.*
- class [LinearGradient](#)
- class [logic\\_error](#)
- class [Matrix](#)  
*A Transformation matrix.*
- class [Path](#)  
*A data structure for holding a path.*
- class [Pattern](#)  
*[Cairo::Pattern](#) is the paint with which cairo draws.*
- class [PdfSurface](#)  
*A [PdfSurface](#) provides a way to render PDF documents from cairo.*
- class [PsSurface](#)  
*A [PsSurface](#) provides a way to render PostScript documents from cairo.*
- class [QuartzFontFace](#)  
*The Quartz font backend is primarily used to render text on Apple MacOS X systems.*

- class [QuartzSurface](#)  
A [QuartzSurface](#) provides a way to render within Apple Mac OS X.
- class [RadialGradient](#)
- class [RecordingSurface](#)  
A recording surface is a surface that records all drawing operations at the highest level of the surface backend interface, (that is, the level of paint, mask, stroke, fill, and `show_text_glyphs`).
- class [Region](#)  
A simple graphical data type representing an area of integer-aligned rectangles.
- class [SaveGuard](#)  
RAII-style context save/restore class.
- class [ScaledFont](#)  
A [ScaledFont](#) is a font scaled to a particular size and device resolution.
- class [SolidPattern](#)
- class [Surface](#)  
A cairo surface represents an image, either as the destination of a drawing operation or as source when drawing onto another surface.
- class [SurfacePattern](#)
- class [SvgSurface](#)  
A [SvgSurface](#) provides a way to render Scalable Vector Graphics (SVG) images from cairo.
- class [ToyFontFace](#)  
A simple font face used for the cairo 'toy' font API.
- class [UserFontFace](#)  
Font support with font data provided by the user.
- class [Win32FontFace](#)  
Font support for Microsoft Windows.
- class [Win32PrintingSurface](#)  
A multi-page vector surface type for printing on Microsoft Windows.
- class [Win32ScaledFont](#)  
Scaled Font implementation for Microsoft Windows fonts.
- class [Win32Surface](#)  
A [Win32Surface](#) provides a way to render within Microsoft Windows.
- class [XlibSurface](#)  
An [XlibSurface](#) provides a way to render to the X Window System using XLib.

## Typedefs

- `template<typename T_CppObject >`  
using [RefPtr](#) = `std::shared_ptr<T_CppObject>`  
*RefPtr<> is a reference-counting shared smartpointer.*
- `typedef cairo_rectangle_t` [Rectangle](#)  
See the [cairo\\_rectangle\\_t reference](#) in the cairo manual for more information.
- `typedef cairo_rectangle_int_t` [RectangleInt](#)  
See the [cairo\\_rectangle\\_int\\_t reference](#) in the cairo manual for more information.
- `typedef cairo_font_extents_t` [FontExtents](#)  
See the [cairo\\_font\\_extents\\_t reference](#) in the cairo manual for more information.
- `typedef cairo_text_extents_t` [TextExtents](#)  
See the [cairo\\_text\\_extents\\_t reference](#) in the cairo manual for more information.
- `typedef cairo_text_cluster_t` [TextCluster](#)  
See the [cairo\\_text\\_cluster\\_t reference](#) in the cairo manual for more information.
- `typedef cairo_glyph_t` [Glyph](#)  
See the [cairo\\_glyph\\_t reference](#) in the cairo manual for more information.

## Enumerations

- enum [Antialias](#) {  
[ANTIALIAS\\_DEFAULT](#) = CAIRO\_ANTIALIAS\_DEFAULT ,  
[ANTIALIAS\\_NONE](#) = CAIRO\_ANTIALIAS\_NONE ,  
[ANTIALIAS\\_GRAY](#) = CAIRO\_ANTIALIAS\_GRAY ,  
[ANTIALIAS\\_SUBPIXEL](#) = CAIRO\_ANTIALIAS\_SUBPIXEL }  
*Specifies the type of antialiasing to do when rendering text or shapes.*
- enum [Content](#) {  
[CONTENT\\_COLOR](#) = CAIRO\_CONTENT\_COLOR ,  
[CONTENT\\_ALPHA](#) = CAIRO\_CONTENT\_ALPHA ,  
[CONTENT\\_COLOR\\_ALPHA](#) = CAIRO\_CONTENT\_COLOR\_ALPHA }  
*[Cairo::Content](#) is used to describe the content that a surface will contain, whether color information, alpha information (translucence vs.*
- enum [SubpixelOrder](#) {  
[SUBPIXEL\\_ORDER\\_DEFAULT](#) = CAIRO\_SUBPIXEL\_ORDER\_DEFAULT ,  
[SUBPIXEL\\_ORDER\\_RGB](#) = CAIRO\_SUBPIXEL\_ORDER\_RGB ,  
[SUBPIXEL\\_ORDER\\_BGR](#) = CAIRO\_SUBPIXEL\_ORDER\_BGR ,  
[SUBPIXEL\\_ORDER\\_VRGB](#) = CAIRO\_SUBPIXEL\_ORDER\_VRGB ,  
[SUBPIXEL\\_ORDER\\_VBGR](#) = CAIRO\_SUBPIXEL\_ORDER\_VBGR }  
*The subpixel order specifies the order of color elements within each pixel on the display device when rendering with an antialiasing mode of [Cairo::ANTIALIAS\\_SUBPIXEL](#).*
- enum [FontType](#) {  
[FONT\\_TYPE\\_TOY](#) = CAIRO\_FONT\_TYPE\_TOY ,  
[FONT\\_TYPE\\_FT](#) = CAIRO\_FONT\_TYPE\_FT ,  
[FONT\\_TYPE\\_WIN32](#) = CAIRO\_FONT\_TYPE\_WIN32 ,  
[FONT\\_TYPE\\_QUARTZ](#) = CAIRO\_FONT\_TYPE\_QUARTZ ,  
[FONT\\_TYPE\\_USER](#) = CAIRO\_FONT\_TYPE\_USER }  
*[Cairo::FontType](#) is used to describe the type of a given font face or scaled font.*
- enum [TextClusterFlags](#) { [TEXT\\_CLUSTER\\_FLAG\\_BACKWARD](#) = CAIRO\_TEXT\_CLUSTER\_FLAG\_↵  
BACKWARD }  
*Specifies properties of a text cluster mapping.*
- enum [FtSynthesize](#) {  
[FT\\_SYNTHESIZE\\_BOLD](#) = CAIRO\_FT\_SYNTHESIZE\_BOLD ,  
[FT\\_SYNTHESIZE\\_OBLIQUE](#) = CAIRO\_FT\_SYNTHESIZE\_OBLIQUE }  
*A set of synthesis options to control how FreeType renders the glyphs for a particular font face.*
- enum [PdfVersion](#) {  
[PDF\\_VERSION\\_1\\_4](#) = CAIRO\_PDF\_VERSION\_1\_4 ,  
[PDF\\_VERSION\\_1\\_5](#) = CAIRO\_PDF\_VERSION\_1\_5 }
- enum [PsLevel](#) {  
[PS\\_LEVEL\\_2](#) = CAIRO\_PS\_LEVEL\_2 ,  
[PS\\_LEVEL\\_3](#) = CAIRO\_PS\_LEVEL\_3 }  
*describes the language level of the PostScript Language Reference that a generated PostScript file will conform to.*
- enum [SvgVersion](#) {  
[SVG\\_VERSION\\_1\\_1](#) = CAIRO\_SVG\_VERSION\_1\_1 ,  
[SVG\\_VERSION\\_1\\_2](#) = CAIRO\_SVG\_VERSION\_1\_2 }

## Functions

- [FtSynthesize operator|](#) ([FtSynthesize](#) a, [FtSynthesize](#) b)
- [FtSynthesize operator&](#) ([FtSynthesize](#) a, [FtSynthesize](#) b)
- template<typename T\_CppObject >  
[RefPtr](#)< T\_CppObject > [make\\_refptr\\_for\\_instance](#) (T\_CppObject \*object)  
*Create a [RefPtr](#)<> to an instance of any reference-counted class whose destructor is noexcept (the default for destructors).*

## 7.1.1 Typedef Documentation

### 7.1.1.1 FontExtents

```
typedef cairo_font_extents_t Cairo::FontExtents
```

See the [cairo\\_font\\_extents\\_t reference](#) in the cairo manual for more information.

### 7.1.1.2 Glyph

```
typedef cairo_glyph_t Cairo::Glyph
```

See the [cairo\\_glyph\\_t reference](#) in the cairo manual for more information.

### 7.1.1.3 Rectangle

```
typedef cairo_rectangle_t Cairo::Rectangle
```

See the [cairo\\_rectangle\\_t reference](#) in the cairo manual for more information.

### 7.1.1.4 RectangleInt

```
typedef cairo_rectangle_int_t Cairo::RectangleInt
```

See the [cairo\\_rectangle\\_int\\_t reference](#) in the cairo manual for more information.

### 7.1.1.5 RefPtr

```
template<typename T_CppObject >  
using Cairo::RefPtr = std::shared_ptr<T_CppObject>
```

RefPtr<> is a reference-counting shared smartpointer.

Reference counting means that a shared reference count is incremented each time a RefPtr is copied, and decremented each time a RefPtr is destroyed, for instance when it leaves its scope. When the reference count reaches zero, the contained object is deleted

caiomm uses RefPtr so that you don't need to remember to delete the object explicitly, or know when a method expects you to delete the object that it returns, and to prevent any need to manually reference and unreference cairo objects.

See also

[Cairo::make\\_refptr\\_for\\_instance\(\)](#)

### 7.1.1.6 TextCluster

```
typedef cairo_text_cluster_t Cairo::TextCluster
```

See the [cairo\\_text\\_cluster\\_t reference](#) in the cairo manual for more information.

### 7.1.1.7 TextExtents

```
typedef cairo_text_extents_t Cairo::TextExtents
```

See the [cairo\\_text\\_extents\\_t reference](#) in the cairo manual for more information.

## 7.1.2 Enumeration Type Documentation

### 7.1.2.1 Antialias

```
enum Cairo::Antialias
```

Specifies the type of antialiasing to do when rendering text or shapes.

The interpretation of [Cairo::ANTIALIAS\\_DEFAULT](#) is left entirely up to the backend.

Enumerator

ANTIALIAS_DEFAULT	Use the default antialiasing for the subsystem and target device.
ANTIALIAS_NONE	Use bilevel alpha mask.
ANTIALIAS_GRAY	Perform single-color antialiasing (using shades of gray for black text on white background, for example).
ANTIALIAS_SUBPIXEL	Perform antialiasing by taking advantage of the order of subpixel elements on devices such as LCD panels.

### 7.1.2.2 Content

```
enum Cairo::Content
```

[Cairo::Content](#) is used to describe the content that a surface will contain, whether color information, alpha information (translucence vs.

opacity), or both.

Enumerator

CONTENT_COLOR	The surface will hold color content only.
CONTENT_ALPHA	The surface will hold alpha content only.
CONTENT_COLOR_ALPHA	The surface will hold color and alpha content.

### 7.1.2.3 FontType

```
enum Cairo::FontType
```

[Cairo::FontType](#) is used to describe the type of a given font face or scaled font.

The font types are also known as "font backends" within cairo.

New entries may be added in future versions.

Since

1.2

## Enumerator

FONT_TYPE_TOY	The font was created using cairo's toy font api.
FONT_TYPE_FT	The font is of type FreeType.
FONT_TYPE_WIN32	The font is of type Win32.
FONT_TYPE_QUARTZ	The font is of type Quartz.  Since 1.6
FONT_TYPE_USER	The font was created using cairo's user font api.  Since 1.8

## 7.1.2.4 FtSynthesize

```
enum Cairo::FtSynthesize
```

A set of synthesis options to control how FreeType renders the glyphs for a particular font face.

FreeType provides the ability to synthesize different glyphs from a base font, which is useful if you lack those glyphs from a true bold or oblique font.

Individual synthesis features of a `FtFontFace` can be set using `FtFontFace::set_synthesize()`, or disabled using `FtFontFace::unset_synthesize()`. The currently enabled set of synthesis options can be queried with `FtFontFace::get_synthesize()`.

Note: that when synthesizing glyphs, the font metrics returned will only be estimates.

Since

1.12

## Enumerator

FT_SYNTHESIZE_BOLT	Embolden the glyphs (redraw with a pixel offset)
FT_SYNTHESIZE_OBLIQUE	Slant the glyph outline by 12 degrees to the right.

## 7.1.2.5 PdfVersion

```
enum Cairo::PdfVersion
```

## Enumerator

PDF_VERSION_1↔ _4	
PDF_VERSION_1↔ _5	

## 7.1.2.6 PsLevel

```
enum Cairo::PsLevel
```

describes the language level of the PostScript Language Reference that a generated PostScript file will conform to.

## Enumerator

PS_LEVEL↔ _2	
PS_LEVEL↔ _3	

## 7.1.2.7 SubpixelOrder

```
enum Cairo::SubpixelOrder
```

The subpixel order specifies the order of color elements within each pixel on the display device when rendering with an antialiasing mode of [Cairo::ANTIALIAS\\_SUBPIXEL](#).

## Enumerator

SUBPIXEL_ORDER_DEFAULT	Use the default subpixel order for for the target device.
SUBPIXEL_ORDER_RGB	Subpixel elements are arranged horizontally with red at the left.
SUBPIXEL_ORDER_BGR	Subpixel elements are arranged horizontally with blue at the left.
SUBPIXEL_ORDER_VRGB	Subpixel elements are arranged vertically with red at the top.
SUBPIXEL_ORDER_VBGR	Subpixel elements are arranged vertically with blue at the top.

## 7.1.2.8 SvgVersion

```
enum Cairo::SvgVersion
```

## Enumerator

SVG_VERSION_1↔ _1	
SVG_VERSION_1↔ _2	

## 7.1.2.9 TextClusterFlags

```
enum Cairo::TextClusterFlags
```

Specifies properties of a text cluster mapping.

## Since

1.8

## Enumerator

TEXT_CLUSTER_FLAG_BACKWARD	The clusters in the cluster array map to glyphs in the glyph array from end to start.
----------------------------	---------------------------------------------------------------------------------------

### 7.1.3 Function Documentation

#### 7.1.3.1 `make_refptr_for_instance()`

```
template<typename T_CppObject >
RefPtr< T_CppObject > Cairo::make_refptr_for_instance (
    T_CppObject * object)
```

Create a `RefPtr<>` to an instance of any reference-counted class whose destructor is `noexcept` (the default for destructors).

Normal application code should not need to use this. However, this is necessary when implementing `create()` methods for derived classes, such as a derived [Cairo::UserFontFace](#).

#### Examples

[user-font.cc](#).

#### 7.1.3.2 `operator&()`

```
FtSynthesize Cairo::operator& (
    FtSynthesize a,
    FtSynthesize b) [inline]
```

#### 7.1.3.3 `operator" | ()`

```
FtSynthesize Cairo::operator| (
    FtSynthesize a,
    FtSynthesize b) [inline]
```



# Chapter 8

## Class Documentation

### 8.1 Cairo::ColorStop Struct Reference

```
#include <cairomm/pattern.h>
```

#### Public Attributes

- double [offset](#)
- double [red](#)
- double [green](#)
- double [blue](#)
- double [alpha](#)

#### 8.1.1 Member Data Documentation

##### 8.1.1.1 [alpha](#)

```
double Cairo::ColorStop::alpha
```

##### 8.1.1.2 [blue](#)

```
double Cairo::ColorStop::blue
```

##### 8.1.1.3 [green](#)

```
double Cairo::ColorStop::green
```

##### 8.1.1.4 [offset](#)

```
double Cairo::ColorStop::offset
```

### 8.1.1.5 red

```
double Cairo::ColorStop::red
```

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

- cairomm/pattern.h

## 8.2 Cairo::Context Class Reference

[Context](#) is the main class used to draw in cairomm.

```
#include <cairomm/context.h>
```

### Public Types

- enum class [Operator](#) {  
[CLEAR](#) = CAIRO\_OPERATOR\_CLEAR ,  
[SOURCE](#) = CAIRO\_OPERATOR\_SOURCE ,  
[OVER](#) = CAIRO\_OPERATOR\_OVER ,  
[IN](#) = CAIRO\_OPERATOR\_IN ,  
[OUT](#) = CAIRO\_OPERATOR\_OUT ,  
[ATOP](#) = CAIRO\_OPERATOR\_ATOP ,  
[DEST](#) = CAIRO\_OPERATOR\_DEST ,  
[DEST\\_OVER](#) = CAIRO\_OPERATOR\_DEST\_OVER ,  
[DEST\\_IN](#) = CAIRO\_OPERATOR\_DEST\_IN ,  
[DEST\\_OUT](#) = CAIRO\_OPERATOR\_DEST\_OUT ,  
[DEST\\_ATOP](#) = CAIRO\_OPERATOR\_DEST\_ATOP ,  
[XOR](#) = CAIRO\_OPERATOR\_XOR ,  
[ADD](#) = CAIRO\_OPERATOR\_ADD ,  
[SATURATE](#) = CAIRO\_OPERATOR\_SATURATE }  
*Operator is used to set the compositing operator for all cairo drawing operations.*
- enum class [FillRule](#) {  
[WINDING](#) = CAIRO\_FILL\_RULE\_WINDING ,  
[EVEN\\_ODD](#) = CAIRO\_FILL\_RULE\_EVEN\_ODD }  
*FillRule is used to select how paths are filled.*
- enum class [LineCap](#) {  
[BUTT](#) = CAIRO\_LINE\_CAP\_BUTT ,  
[ROUND](#) = CAIRO\_LINE\_CAP\_ROUND ,  
[SQUARE](#) = CAIRO\_LINE\_CAP\_SQUARE }  
*Specifies how to render the endpoints of the path when stroking.*
- enum class [LineJoin](#) {  
[MITER](#) = CAIRO\_LINE\_JOIN\_MITER ,  
[ROUND](#) = CAIRO\_LINE\_JOIN\_ROUND ,  
[BEVEL](#) = CAIRO\_LINE\_JOIN\_BEVEL }  
*Specifies how to render the junction of two lines when stroking.*
- typedef cairo\_t [cobject](#)  
*The base cairo C type that is wrapped by [Cairo::Context](#).*

## Public Member Functions

- [Context](#) (cairo\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Context](#) (const [Context](#) &)=delete
- [Context](#) & [operator=](#) (const [Context](#) &)=delete
- virtual [~Context](#) ()
- void [save](#) ()  
*Makes a copy of the current state of the [Context](#) and saves it on an internal stack of saved states.*
- void [restore](#) ()  
*Restores cr to the state saved by a preceding call to [save\(\)](#) and removes that state from the stack of saved states.*
- void [set\\_operator](#) ([Operator](#) op)  
*Sets the compositing operator to be used for all drawing operations.*
- void [set\\_source](#) (const [RefPtr](#)< const [Pattern](#) > &source)  
*Sets the source pattern within the [Context](#) to source.*
- void [set\\_source\\_rgb](#) (double red, double green, double blue)  
*Sets the source pattern within the [Context](#) to an opaque color.*
- void [set\\_source\\_rgba](#) (double red, double green, double blue, double alpha)  
*Sets the source pattern within the [Context](#) to a translucent color.*
- void [set\\_source](#) (const [RefPtr](#)< [Surface](#) > &surface, double x, double y)  
*This is a convenience function for creating a pattern from a [Surface](#) and setting it as the source.*
- void [set\\_tolerance](#) (double tolerance)  
*Sets the tolerance used when converting paths into trapezoids.*
- void [set\\_antialias](#) ([Antialias](#) antialias)  
*Set the antialiasing mode of the rasterizer used for drawing shapes.*
- void [set\\_fill\\_rule](#) ([FillRule](#) fill\_rule)  
*Set the current fill rule within the cairo [Context](#).*
- void [set\\_line\\_width](#) (double width)  
*Sets the current line width within the cairo [Context](#).*
- void [set\\_line\\_cap](#) ([LineCap](#) line\_cap)  
*Sets the current line cap style within the cairo [Context](#).*
- void [set\\_line\\_join](#) ([LineJoin](#) line\_join)  
*Sets the current line join style within the cairo [Context](#).*
- void [set\\_dash](#) (const [std::valarray](#)< double > &dashes, double offset)  
*Alternate version of [set\\_dash\(\)](#).*
- void [set\\_dash](#) (const [std::vector](#)< double > &dashes, double offset)  
*Sets the dash pattern to be used by [stroke\(\)](#).*
- void [unset\\_dash](#) ()  
*This function disables a dash pattern that was set with [set\\_dash\(\)](#)*
- void [set\\_miter\\_limit](#) (double limit)  
*Sets the current miter limit within the cairo context.*
- void [translate](#) (double tx, double ty)  
*Modifies the current transformation matrix (CTM) by translating the user-space origin by (tx, ty).*
- void [scale](#) (double sx, double sy)  
*Modifies the current transformation matrix (CTM) by scaling the X and Y user-space axes by sx and sy respectively.*
- void [rotate](#) (double angle\_radians)  
*Modifies the current transformation matrix (CTM) by rotating the user-space axes by angle radians.*
- void [rotate\\_degrees](#) (double angle\_degrees)  
*A convenience wrapper around [rotate\(\)](#) that accepts angles in degrees.*
- void [transform](#) (const [Matrix](#) &matrix)  
*Modifies the current transformation matrix (CTM) by applying matrix as an additional transformation.*

- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Modifies the current transformation matrix (CTM) by setting it equal to matrix.*
- void [set\\_identity\\_matrix](#) ()  
*Resets the current transformation matrix (CTM) by setting it equal to the identity matrix.*
- void [user\\_to\\_device](#) (double &x, double &y) const  
*Transform a coordinate from user space to device space by multiplying the given point by the current transformation matrix (CTM).*
- void [user\\_to\\_device\\_distance](#) (double &dx, double &dy) const  
*Transform a distance vector from user space to device space.*
- void [device\\_to\\_user](#) (double &x, double &y) const  
*Transform a coordinate from device space to user space by multiplying the given point by the inverse of the current transformation matrix (CTM).*
- void [device\\_to\\_user\\_distance](#) (double &dx, double &dy) const  
*Transform a distance vector from device space to user space.*
- void [begin\\_new\\_path](#) ()  
*Clears the current path.*
- void [begin\\_new\\_sub\\_path](#) ()  
*Begin a new subpath.*
- void [move\\_to](#) (double x, double y)  
*If the current subpath is not empty, begin a new subpath.*
- void [line\\_to](#) (double x, double y)  
*Adds a line to the path from the current point to position (x, y) in user-space coordinates.*
- void [curve\\_to](#) (double x1, double y1, double x2, double y2, double x3, double y3)  
*Adds a cubic Bezier spline to the path from the current point to position (x3, y3) in user-space coordinates, using (x1, y1) and (x2, y2) as the control points.*
- void [arc](#) (double xc, double yc, double radius, double angle1, double angle2)  
*Adds a circular arc of the given radius to the current path.*
- void [arc\\_negative](#) (double xc, double yc, double radius, double angle1, double angle2)  
*Adds a circular arc of the given radius to the current path.*
- void [rel\\_move\\_to](#) (double dx, double dy)  
*If the current subpath is not empty, begin a new subpath.*
- void [rel\\_line\\_to](#) (double dx, double dy)  
*Relative-coordinate version of [line\\_to](#)().*
- void [rel\\_curve\\_to](#) (double dx1, double dy1, double dx2, double dy2, double dx3, double dy3)  
*Relative-coordinate version of [curve\\_to](#)().*
- void [rectangle](#) (double x, double y, double width, double height)  
*Adds a closed-subpath rectangle of the given size to the current path at position (x, y) in user-space coordinates.*
- void [close\\_path](#) ()  
*Adds a line segment to the path from the current point to the beginning of the current subpath, (the most recent point passed to [move\\_to](#)()), and closes this subpath.*
- void [paint](#) ()  
*A drawing operator that paints the current source everywhere within the current clip region.*
- void [paint\\_with\\_alpha](#) (double alpha)  
*A drawing operator that paints the current source everywhere within the current clip region using a mask of constant alpha value alpha.*
- void [mask](#) (const [RefPtr](#)< const [Pattern](#) > &pattern)  
*A drawing operator that paints the current source using the alpha channel of pattern as a mask.*
- void [mask](#) (const [RefPtr](#)< const [Surface](#) > &surface, double surface\_x, double surface\_y)  
*A drawing operator that paints the current source using the alpha channel of surface as a mask.*
- void [stroke](#) ()  
*A drawing operator that strokes the current [Path](#) according to the current line width, line join, line cap, and dash settings.*

- void [stroke\\_preserve](#) ()  
*A drawing operator that strokes the current [Path](#) according to the current line width, line join, line cap, and dash settings.*
- void [fill](#) ()  
*A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled).*
- void [fill\\_preserve](#) ()  
*A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled).*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so, the contents of the current page will be retained for the next page too.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [in\\_stroke](#) (double x, double y) const  
*Tests whether the given point is inside the area that would be affected by a [stroke\(\)](#) operation given the current path and stroking parameters.*
- bool [in\\_fill](#) (double x, double y) const  
*Tests whether the given point is inside the area that would be affected by a [fill\(\)](#) operation given the current path and filling parameters.*
- bool [in\\_clip](#) (double x, double y) const  
*Tests whether the given point is inside the area that would be visible through the current clip, i.e.*
- void [get\\_stroke\\_extents](#) (double &x1, double &y1, double &x2, double &y2) const  
*Computes a bounding box in user coordinates covering the area that would be affected, (the "inked" area), by a [stroke\(\)](#) operation given the current path and stroke parameters.*
- void [get\\_fill\\_extents](#) (double &x1, double &y1, double &x2, double &y2) const  
*Computes a bounding box in user coordinates covering the area that would be affected, (the "inked" area), by a [fill\(\)](#) operation given the current path and fill parameters.*
- void [reset\\_clip](#) ()  
*Reset the current clip region to its original, unrestricted state.*
- void [clip](#) ()  
*Establishes a new clip region by intersecting the current clip region with the current [Path](#) as it would be filled by [fill\(\)](#) and according to the current fill rule.*
- void [clip\\_preserve](#) ()  
*Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by [fill\(\)](#) and according to the current fill rule.*
- void [get\\_clip\\_extents](#) (double &x1, double &y1, double &x2, double &y2) const  
*Computes a bounding box in user coordinates covering the area inside the current clip.*
- void [copy\\_clip\\_rectangle\\_list](#) ( **std::vector**< [Rectangle](#) > &rectangles) const  
*Returns the current clip region as a list of rectangles in user coordinates.*
- void [select\\_font\\_face](#) (const **std::string** &family, [ToyFontFace::Slant](#) slant, [ToyFontFace::Weight](#) weight)  
*Selects a family and style of font from a simplified description as a family name, slant and weight.*
- void [set\\_font\\_size](#) (double **size**)  
*Sets the current font matrix to a scale by a factor of size, replacing any font matrix previously set with [set\\_font\\_size\(\)](#) or [set\\_font\\_matrix\(\)](#).*
- void [set\\_font\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the current font matrix to @matrix.*
- void [get\\_font\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the current font matrix.*
- void [set\\_font\\_options](#) (const [FontOptions](#) &options)  
*Sets a set of custom font rendering options.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves font rendering options set via [set\\_font\\_options\(\)](#).*

- void [set\\_scaled\\_font](#) (const [RefPtr](#)< const [ScaledFont](#) > &scaled\_font)  
*Replaces the current font face, font matrix, and font options in the context with those of the scaled\_font.*
- [RefPtr](#)< [ScaledFont](#) > [get\\_scaled\\_font](#) ()  
*Gets the current scaled font.*
- void [show\\_text](#) (const **std::string** &utf8)  
*A drawing operator that generates the shape from a string of UTF-8 characters, rendered according to the current font\_face, font\_size (font\_matrix), and font\_options.*
- void [show\\_glyphs](#) (const **std::vector**< [Glyph](#) > &glyphs)  
*A drawing operator that generates the shape from an array of glyphs, rendered according to the current font face, font size (font\_matrix), and font\_options.*
- void [show\\_text\\_glyphs](#) (const **std::string** &utf8, const **std::vector**< [Glyph](#) > &glyphs, const **std::vector**< [TextCluster](#) > &clusters, [TextClusterFlags](#) cluster\_flags)  
*This operation has rendering effects similar to [show\\_glyphs\(\)](#) but, if the target surface supports it, uses the provided text and cluster mapping to embed the text for the glyphs shown in the output.*
- void [get\\_font\\_extents](#) ([FontExtents](#) &extents) const  
*Gets the font extents for the currently selected font.*
- void [set\\_font\\_face](#) (const [RefPtr](#)< const [FontFace](#) > &font\_face)  
*Replaces the current font face in the context with font\_face font\_face.*
- void [get\\_text\\_extents](#) (const **std::string** &utf8, [TextExtents](#) &extents) const  
*Gets the extents for a string of text.*
- void [get\\_glyph\\_extents](#) (const **std::vector**< [Glyph](#) > &glyphs, [TextExtents](#) &extents) const  
*Gets the extents for an array of glyphs.*
- void [text\\_path](#) (const **std::string** &utf8)  
*Adds closed paths for text to the current path.*
- void [glyph\\_path](#) (const **std::vector**< [Glyph](#) > &glyphs)  
*Adds closed paths for the glyphs to the current path.*
- [Operator](#) [get\\_operator](#) () const  
*Gets the current compositing operator for a cairo [Context](#).*
- double [get\\_tolerance](#) () const  
*Gets the current tolerance value, as set by [set\\_tolerance\(\)](#)*
- [Antialias](#) [get\\_antialias](#) () const  
*Gets the current shape antialiasing mode, as set by [set\\_antialias\(\)](#)*
- void [get\\_current\\_point](#) (double &x, double &y) const  
*Gets the current point of the current path, which is conceptually the final point reached by the path so far.*
- bool [has\\_current\\_point](#) () const  
*Checks if there is a current point defined.*
- [FillRule](#) [get\\_fill\\_rule](#) () const  
*Gets the current fill rule, as set by [set\\_fill\\_rule\(\)](#).*
- double [get\\_line\\_width](#) () const  
*Gets the current line width, as set by [set\\_line\\_width\(\)](#).*
- [LineCap](#) [get\\_line\\_cap](#) () const  
*Gets the current line cap style, as set by [set\\_line\\_cap\(\)](#)*
- [LineJoin](#) [get\\_line\\_join](#) () const  
*Gets the current line join style, as set by [set\\_line\\_join\(\)](#)*
- double [get\\_miter\\_limit](#) () const  
*Gets the current miter limit, as set by [set\\_miter\\_limit\(\)](#)*
- void [get\\_dash](#) (**std::vector**< double > &dashes, double &offset) const  
*Gets the current dash array and offset.*
- void [get\\_matrix](#) ([Matrix](#) &matrix)  
*Stores the current transformation matrix (CTM) into matrix.*
- [Matrix](#) [get\\_matrix](#) () const

- Returns the current transformation matrix (CTM)*

    - [Path](#) \* [copy\\_path](#) () const

*Creates a copy of the current path and returns it to the user.*
  - void [get\\_path\\_extents](#) (double &x1, double &y1, double &x2, double &y2) const

*Computes a bounding box in user-space coordinates covering the points on the current path.*
  - [Path](#) \* [copy\\_path\\_flat](#) () const

*Gets a flattened copy of the current path and returns it to the user.*
  - void [append\\_path](#) (const [Path](#) &path)

*Append the path onto the current path.*
  - void [push\\_group](#) ()

*Temporarily redirects drawing to an intermediate surface known as a group.*
  - void [push\\_group\\_with\\_content](#) ([Content](#) content)

*Temporarily redirects drawing to an intermediate surface known as a group.*
  - [RefPtr](#)< [Pattern](#) > [pop\\_group](#) ()

*Terminates the redirection begun by a call to [push\\_group\(\)](#) or [push\\_group\\_with\\_content\(\)](#) and returns a new pattern containing the results of all drawing operations performed to the group.*
  - void [pop\\_group\\_to\\_source](#) ()

*Terminates the redirection begun by a call to [push\\_group\(\)](#) or [push\\_group\\_with\\_content\(\)](#) and installs the resulting pattern as the source pattern in the given cairo [Context](#).*
  - [RefPtr](#)< [Surface](#) > [get\\_group\\_target](#) ()

*Gets the target surface for the current group as started by the most recent call to [push\\_group\(\)](#) or [push\\_group\\_with\\_content\(\)](#).*
  - [RefPtr](#)< const [Surface](#) > [get\\_group\\_target](#) () const

*Same as the non-const version but returns a reference to a const [Surface](#).*
  - [cobject](#) \* [cobj](#) ()

*Gets a pointer to the base C type that is wrapped by the [Context](#).*
  - const [cobject](#) \* [cobj](#) () const

*Gets a pointer to the base C type that is wrapped by the [Context](#).*
- 
- [RefPtr](#)< [FontFace](#) > [get\\_font\\_face](#) ()

*Gets the current font face.*
  - [RefPtr](#)< const [FontFace](#) > [get\\_font\\_face](#) () const
- 
- [RefPtr](#)< [Pattern](#) > [get\\_source](#) ()

*Gets the current source pattern for the [Context](#).*
  - [RefPtr](#)< const [Pattern](#) > [get\\_source](#) () const
  - [RefPtr](#)< [SurfacePattern](#) > [get\\_source\\_for\\_surface](#) ()

*Gets the current source surface pattern for the [Context](#), if any.*
  - [RefPtr](#)< const [SurfacePattern](#) > [get\\_source\\_for\\_surface](#) () const
- 
- [RefPtr](#)< [Surface](#) > [get\\_target](#) ()

*Gets the target surface associated with this [Context](#).*
  - [RefPtr](#)< const [Surface](#) > [get\\_target](#) () const

### Static Public Member Functions

- static [RefPtr< Context > create](#) (const [RefPtr< Surface >](#) &target)

### Protected Member Functions

- [Context](#) (const [RefPtr< Surface >](#) &target)

### Protected Attributes

- [cobject](#) \* [m\\_cobject](#)

## 8.2.1 Detailed Description

[Context](#) is the main class used to draw in cairomm.

It contains the current state of the rendering device, including coordinates of yet to be drawn shapes.

In the simplest case, create a [Context](#) with its target [Surface](#), set its drawing options (line width, color, etc), create shapes with methods like [move\\_to\(\)](#) and [line\\_to\(\)](#), and then draw the shapes to the [Surface](#) using methods such as [stroke\(\)](#) or [fill\(\)](#).

[Context](#) is a reference-counted object that should be used via [Cairo::RefPtr](#).

## 8.2.2 Member Typedef Documentation

### 8.2.2.1 cobject

```
typedef cairo_t Cairo::Context::cobject
```

The base cairo C type that is wrapped by [Cairo::Context](#).

## 8.2.3 Member Enumeration Documentation

### 8.2.3.1 FillRule

```
enum class Cairo::Context::FillRule [strong]
```

[FillRule](#) is used to select how paths are filled.

For both fill rules, whether or not a point is included in the fill is determined by taking a ray from that point to infinity and looking at intersections with the path. The ray can be in any direction, as long as it doesn't pass through the end point of a segment or have a tricky intersection such as intersecting tangent to the path. (Note that filling is not actually implemented in this way. This is just a description of the rule that is applied.)

The default fill rule is [Cairo::FillRule::WINDING](#).

New entries may be added in future versions.



## Enumerator

WINDING	If the path crosses the ray from left-to-right, counts +1. If the path crosses the ray from right to left, counts -1. (Left and right are determined from the perspective of looking along the ray from the starting point.) If the total count is non-zero, the point will be filled.
EVEN_ODD	Counts the total number of intersections, without regard to the orientation of the contour. If the total number of intersections is odd, the point will be filled.

## 8.2.3.2 LineCap

```
enum class Cairo::Context::LineCap [strong]
```

Specifies how to render the endpoints of the path when stroking.

The default line cap style is Cairo::LineCap::BUTT.

## Enumerator

BUTT	Start(stop) the line exactly at the start(end) point.
ROUND	Use a round ending, the center of the circle is the end point.
SQUARE	Use a squared ending, the center of the square is the end point.

## 8.2.3.3 LineJoin

```
enum class Cairo::Context::LineJoin [strong]
```

Specifies how to render the junction of two lines when stroking.

The default line join style is Cairo::LineJoin::MITER.

## Enumerator

MITER	Use a sharp (angled) corner, see <a href="#">Context::set_miter_limit()</a>
ROUND	Use a rounded join, the center of the circle is the joint point.
BEVEL	Use cut-off join, the join is cut off at half the line width from the join point.

## 8.2.3.4 Operator

```
enum class Cairo::Context::Operator [strong]
```

Operator is used to set the compositing operator for all cairo drawing operations.

The default operator is Cairo::Operator::OVER.

The operators marked as *unbounded* modify their destination even outside of the mask layer (that is, their effect is not bound by the mask layer). However, their effect can still be limited by way of clipping.

To keep things simple, the operator descriptions here document the behavior for when both source and destination are either fully transparent or fully opaque. The actual implementation works for translucent layers too. For a more detailed explanation of the effects of each operator, including the mathematical definitions, see [this](#)

## Enumerator

CLEAR	Clear destination layer (bounded)
SOURCE	Replace destination layer (bounded)
OVER	Draw source layer on top of destination layer (bounded)
IN	Draw source where there was destination content (unbounded)
OUT	Draw source where there was no destination content (unbounded)
ATOP	Draw source on top of destination content and only there.
DEST	Ignore the source.
DEST_OVER	Draw destination on top of source.
DEST_IN	Leave destination only where there was source content (unbounded)
DEST_OUT	Leave destination only where there was no source content.
DEST_ATOP	Leave destination on top of source content and only there (unbounded)
XOR	Source and destination are shown where there is only one of them.
ADD	Source and destination layers are accumulated.
SATURATE	Like over, but assuming source and dest are disjoint geometries.

## 8.2.4 Constructor & Destructor Documentation

### 8.2.4.1 Context() [1/3]

```
Cairo::Context::Context (
    const RefPtr< Surface > & target) [explicit], [protected]
```

### 8.2.4.2 Context() [2/3]

```
Cairo::Context::Context (
    cairo_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.2.4.3 Context() [3/3]

```
Cairo::Context::Context (
    const Context & ) [delete]
```

#### 8.2.4.4 ~Context()

```
virtual Cairo::Context::~~Context () [virtual]
```

### 8.2.5 Member Function Documentation

#### 8.2.5.1 append\_path()

```
void Cairo::Context::append_path (
    const Path & path)
```

Append the path onto the current path.

The path may be either the return value from one of [copy\\_path\(\)](#) or [copy\\_path\\_flat\(\)](#) or it may be constructed manually.

##### Parameters

<i>path</i>	path to be appended
-------------	---------------------

#### 8.2.5.2 arc()

```
void Cairo::Context::arc (
    double xc,
    double yc,
    double radius,
    double angle1,
    double angle2)
```

Adds a circular arc of the given radius to the current path.

The arc is centered at (*xc*, *yc*), begins at *angle1* and proceeds in the direction of increasing angles to end at *angle2*. If *angle2* is less than *angle1* it will be progressively increased by  $2 * M\_PI$  until it is greater than *angle1*.

If there is a current point, an initial line segment will be added to the path to connect the current point to the beginning of the arc. If this initial line is undesired, it can be avoided by calling [begin\\_new\\_sub\\_path\(\)](#) before calling [arc\(\)](#).

Angles are measured in radians. An angle of 0 is in the direction of the positive X axis (in user-space). An angle of  $M\_PI/2.0$  radians (90 degrees) is in the direction of the positive Y axis (in user-space). Angles increase in the direction from the positive X axis toward the positive Y axis. So with the default transformation matrix, angles increase in a clockwise direction.

( To convert from degrees to radians, use  $degrees * (M\_PI / 180.0)$ . )

This function gives the arc in the direction of increasing angles; see [arc\\_negative\(\)](#) to get the arc in the direction of decreasing angles.

The arc is circular in user-space. To achieve an elliptical arc, you can scale the current transformation matrix by different amounts in the X and Y directions. For example, to draw an ellipse in the box given by x, y, width, height:

```
context->save();
context->translate(x, y);
context->scale(width / 2.0, height / 2.0);
context->arc(0.0, 0.0, 1.0, 0.0, 2 * M_PI);
context->restore();
```

## Parameters

<i>xc</i>	X position of the center of the arc
<i>yc</i>	Y position of the center of the arc
<i>radius</i>	the radius of the arc
<i>angle1</i>	the start angle, in radians
<i>angle2</i>	the end angle, in radians

**8.2.5.3 arc\_negative()**

```
void Cairo::Context::arc_negative (
    double xc,
    double yc,
    double radius,
    double angle1,
    double angle2)
```

Adds a circular arc of the given *radius* to the current path.

The arc is centered at (*xc*, *yc*), begins at *angle1* and proceeds in the direction of decreasing angles to end at *angle2*. If *angle2* is greater than *angle1* it will be progressively decreased by  $2 * M\_PI$  until it is greater than *angle1*.

See [arc\(\)](#) for more details. This function differs only in the direction of the arc between the two angles.

## Parameters

<i>xc</i>	X position of the center of the arc
<i>yc</i>	Y position of the center of the arc
<i>radius</i>	the radius of the arc
<i>angle1</i>	the start angle, in radians
<i>angle2</i>	the end angle, in radians

**8.2.5.4 begin\_new\_path()**

```
void Cairo::Context::begin_new_path ()
```

Clears the current path.

After this call there will be no current point.

**8.2.5.5 begin\_new\_sub\_path()**

```
void Cairo::Context::begin_new_sub_path ()
```

Begin a new subpath.

Note that the existing path is not affected. After this call there will be no current point.

In many cases, this call is not needed since new subpaths are frequently started with [move\\_to\(\)](#).

A call to [begin\\_new\\_sub\\_path\(\)](#) is particularly useful when beginning a new subpath with one of the [arc\(\)](#) calls. This makes things easier as it is no longer necessary to manually compute the arc's initial coordinates for a call to [move\\_to\(\)](#).

Since

1.2

### 8.2.5.6 clip()

```
void Cairo::Context::clip ()
```

Establishes a new clip region by intersecting the current clip region with the current [Path](#) as it would be filled by [fill\(\)](#) and according to the current fill rule.

After [clip\(\)](#), the current path will be cleared from the cairo [Context](#).

The current clip region affects all drawing operations by effectively masking out any changes to the surface that are outside the current clip region.

Calling [clip\(\)](#) can only make the clip region smaller, never larger. But the current clip is part of the graphics state, so a temporary restriction of the clip region can be achieved by calling [clip\(\)](#) within a [save\(\)/restore\(\)](#) pair. The only other means of increasing the size of the clip region is [reset\\_clip\(\)](#).

See also

[set\\_fill\\_rule\(\)](#)

### 8.2.5.7 clip\_preserve()

```
void Cairo::Context::clip_preserve ()
```

Establishes a new clip region by intersecting the current clip region with the current path as it would be filled by [fill\(\)](#) and according to the current fill rule.

Unlike [clip\(\)](#), [clip\\_preserve](#) preserves the path within the cairo [Context](#).

See also

[clip\(\)](#)

[set\\_fill\\_rule\(\)](#)

### 8.2.5.8 close\_path()

```
void Cairo::Context::close_path ()
```

Adds a line segment to the path from the current point to the beginning of the current subpath, (the most recent point passed to [move\\_to\(\)](#)), and closes this subpath.

After this call the current point will be at the joined endpoint of the sub-path.

The behavior of [close\\_path\(\)](#) is distinct from simply calling [line\\_to\(\)](#) with the equivalent coordinate in the case of stroking. When a closed subpath is stroked, there are no caps on the ends of the subpath. Instead, there is a line join connecting the final and initial segments of the subpath.

If there is no current point before the call to [close\\_path\(\)](#), this function will have no effect.

**8.2.5.9 cobj()** [1/2]

```
cobject * Cairo::Context::cobj () [inline]
```

Gets a pointer to the base C type that is wrapped by the [Context](#).

**8.2.5.10 cobj()** [2/2]

```
const cobject * Cairo::Context::cobj () const [inline]
```

Gets a pointer to the base C type that is wrapped by the [Context](#).

**8.2.5.11 copy\_clip\_rectangle\_list()**

```
void Cairo::Context::copy_clip_rectangle_list (
    std::vector< Rectangle > & rectangles) const
```

Returns the current clip region as a list of rectangles in user coordinates.

This function will throw an exception if the clip region cannot be represented as a list of user-space rectangles.

**Parameters**

<i>rectangles</i>	a vector to store the rectangles into
-------------------	---------------------------------------

**Exceptions**

--	--

**Since**

1.4

**8.2.5.12 copy\_page()**

```
void Cairo::Context::copy_page ()
```

Emits the current page for backends that support multiple pages, but doesn't clear it, so, the contents of the current page will be retained for the next page too.

Use [show\\_page\(\)](#) if you want to get an empty page after the emission.

This is a convenience function that simply calls [Surface::copy\\_page\(\)](#) on *cr*'s target.

### 8.2.5.13 copy\_path()

```
Path * Cairo::Context::copy_path () const
```

Creates a copy of the current path and returns it to the user.

#### Note

The caller owns the [Path](#) object returned from this function. The [Path](#) object must be freed when you are finished with it.

### 8.2.5.14 copy\_path\_flat()

```
Path * Cairo::Context::copy_path_flat () const
```

Gets a flattened copy of the current path and returns it to the user.

This function is like [copy\\_path\(\)](#) except that any curves in the path will be approximated with piecewise-linear approximations, (accurate to within the current tolerance value). That is, the result is guaranteed to not have any elements of type CAIRO\_PATH\_CURVE\_TO which will instead be replaced by a series of CAIRO\_PATH\_LINE\_TO elements.

#### Note

The caller owns the [Path](#) object returned from this function. The [Path](#) object must be freed when you are finished with it.

### 8.2.5.15 create()

```
static RefPtr< Context > Cairo::Context::create (
    const RefPtr< Surface > & target) [static]
```

#### Examples

[image-surface.cc](#), [pdf-surface.cc](#), [ps-surface.cc](#), [svg-surface.cc](#), [toy-text.cc](#), and [user-font.cc](#).

### 8.2.5.16 curve\_to()

```
void Cairo::Context::curve_to (
    double x1,
    double y1,
    double x2,
    double y2,
    double x3,
    double y3)
```

Adds a cubic Bezier spline to the path from the current point to position (x3, y3) in user-space coordinates, using (x1, y1) and (x2, y2) as the control points.

After this call the current point will be (x3, y3).

If there is no current point before the call to [curve\\_to\(\)](#) this function will behave as if preceded by a call to [move\\_to\(x1, y1\)](#).

**Parameters**

<i>x1</i>	the X coordinate of the first control point
<i>y1</i>	the Y coordinate of the first control point
<i>x2</i>	the X coordinate of the second control point
<i>y2</i>	the Y coordinate of the second control point
<i>x3</i>	the X coordinate of the end of the curve
<i>y3</i>	the Y coordinate of the end of the curve

**8.2.5.17 device\_to\_user()**

```
void Cairo::Context::device_to_user (
    double & x,
    double & y) const
```

Transform a coordinate from device space to user space by multiplying the given point by the inverse of the current transformation matrix (CTM).

**Parameters**

<i>x</i>	X value of coordinate (in/out parameter)
<i>y</i>	Y value of coordinate (in/out parameter)

**8.2.5.18 device\_to\_user\_distance()**

```
void Cairo::Context::device_to_user_distance (
    double & dx,
    double & dy) const
```

Transform a distance vector from device space to user space.

This function is similar to [device\\_to\\_user\(\)](#) except that the translation components of the inverse CTM will be ignored when transforming (dx,dy).

**Parameters**

<i>dx</i>	X component of a distance vector (in/out parameter)
<i>dy</i>	Y component of a distance vector (in/out parameter)

**8.2.5.19 fill()**

```
void Cairo::Context::fill ()
```

A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled).

After [fill\(\)](#), the current path will be cleared from the cairo context.

**See also**

[set\\_fill\\_rule\(\)](#)

[fill\\_preserve\(\)](#)



### 8.2.5.20 fill\_preserve()

```
void Cairo::Context::fill_preserve ()
```

A drawing operator that fills the current path according to the current fill rule, (each sub-path is implicitly closed before being filled).

Unlike [fill\(\)](#), [fill\\_preserve\(\)](#) preserves the path within the cairo [Context](#).

See also

[set\\_fill\\_rule\(\)](#)  
[fill\(\)](#).

### 8.2.5.21 get\_antialias()

```
Antialias Cairo::Context::get_antialias () const
```

Gets the current shape antialiasing mode, as set by [set\\_antialias\(\)](#)

### 8.2.5.22 get\_clip\_extents()

```
void Cairo::Context::get_clip_extents (
    double & x1,
    double & y1,
    double & x2,
    double & y2) const
```

Computes a bounding box in user coordinates covering the area inside the current clip.

Parameters

<i>x1</i>	left of the resulting extents
<i>y1</i>	top of the resulting extents
<i>x2</i>	right of the resulting extents
<i>y2</i>	bottom of the resulting extents

Since

1.4

### 8.2.5.23 get\_current\_point()

```
void Cairo::Context::get_current_point (
    double & x,
    double & y) const
```

Gets the current point of the current path, which is conceptually the final point reached by the path so far.

The current point is returned in the user-space coordinate system. If there is no defined current point then x and y will both be set to 0.0. It is possible to check this in advance with [has\\_current\\_point\(\)](#).

Most path construction functions alter the current point. See the following for details on how they affect the current point: [clear\\_path\(\)](#), [move\\_to\(\)](#), [line\\_to\(\)](#), [curve\\_to\(\)](#), [arc\(\)](#), [rel\\_move\\_to\(\)](#), [rel\\_line\\_to\(\)](#), [rel\\_curve\\_to\(\)](#), [arc\(\)](#), and [text\\_path\(\)](#)

Some functions use and alter the current point but do not otherwise change current path: [show\\_text\(\)](#).

Some functions unset the current path and as a result, current point: [fill\(\)](#), [stroke\(\)](#).

## Parameters

<i>x</i>	return value for X coordinate of the current point
<i>y</i>	return value for Y coordinate of the current point

## See also

[has\\_current\\_point\(\)](#)

8.2.5.24 `get_dash()`

```
void Cairo::Context::get_dash (
    std::vector< double > & dashes,
    double & offset) const
```

Gets the current dash array and offset.

## Parameters

<i>dashes</i>	return value for the dash array.
<i>offset</i>	return value for the current dash offset.

## Since

1.4

8.2.5.25 `get_fill_extents()`

```
void Cairo::Context::get_fill_extents (
    double & x1,
    double & y1,
    double & x2,
    double & y2) const
```

Computes a bounding box in user coordinates covering the area that would be affected, (the "inked" area), by a [fill\(\)](#) operation given the current path and fill parameters.

If the current path is empty, returns an empty rectangle ((0,0), (0,0)). [Surface](#) dimensions and clipping are not taken into account.

Contrast with `path_extents()`, which is similar, but returns non-zero extents for some paths with no inked area, (such as a simple line segment).

Note that `fill_extents()` must necessarily do more work to compute the precise inked areas in light of the fill rule, so `path_extents()` may be more desirable for sake of performance if the non-inked path extents are desired.

## Parameters

<i>x1</i>	left of the resulting extents
<i>y1</i>	top of the resulting extents
<i>x2</i>	right of the resulting extents
<i>y2</i>	bottom of the resulting extents

## See also

[fill\(\)](#)

[set\\_fill\\_rule\(\)](#)

[full\\_preserve\(\)](#)

**8.2.5.26 get\_fill\_rule()**

```
FillRule Cairo::Context::get_fill_rule () const
```

Gets the current fill rule, as set by [set\\_fill\\_rule\(\)](#).

**8.2.5.27 get\_font\_extents()**

```
void Cairo::Context::get_font_extents (
    FontExtents & extents) const
```

Gets the font extents for the currently selected font.

**Parameters**

<i>extents</i>	a <a href="#">Cairo::FontExtents</a> object
----------------	---------------------------------------------

**8.2.5.28 get\_font\_face() [1/2]**

```
RefPtr< FontFace > Cairo::Context::get_font_face ()
```

Gets the current font face.

**8.2.5.29 get\_font\_face() [2/2]**

```
RefPtr< const FontFace > Cairo::Context::get_font_face () const
```

**8.2.5.30 get\_font\_matrix()**

```
void Cairo::Context::get_font_matrix (
    Matrix & matrix) const
```

Returns the current font matrix.

**Parameters**

<i>matrix</i>	a <a href="#">Cairo::Matrix</a> to store the results into (in/out parameter)
---------------	------------------------------------------------------------------------------

**See also**

[set\\_font\\_matrix\(\)](#)

**8.2.5.31 get\_font\_options()**

```
void Cairo::Context::get_font_options (
    FontOptions & options) const
```

Retrieves font rendering options set via [set\\_font\\_options\(\)](#).

Note that the returned options do not include any options derived from the underlying surface; they are literally the options passed to [set\\_font\\_options\(\)](#).

## Parameters

<i>options</i>	a <a href="#">FontOptions</a> object into which to store the retrieved options. All existing values are overwritten
----------------	---------------------------------------------------------------------------------------------------------------------

## Since

1.8

**8.2.5.32 get\_glyph\_extents()**

```
void Cairo::Context::get_glyph_extents (
    const std::vector< Glyph > & glyphs,
    TextExtents & extents) const
```

Gets the extents for an array of glyphs.

The extents describe a user-space rectangle that encloses the "inked" portion of the glyphs, (as they would be drawn by [show\\_glyphs\(\)](#)). Additionally, the `x_advance` and `y_advance` values indicate the amount by which the current point would be advanced by [show\\_glyphs\(\)](#).

Note that whitespace glyphs do not contribute to the size of the rectangle (`extents.width` and `extents.height`).

## Parameters

<i>glyphs</i>	a vector of glyphs
<i>extents</i>	a TextExtents object

**8.2.5.33 get\_group\_target()** [1/2]

```
RefPtr< Surface > Cairo::Context::get_group_target ()
```

Gets the target surface for the current group as started by the most recent call to [push\\_group\(\)](#) or [push\\_group\\_with\\_content\(\)](#).

This function will return NULL if called "outside" of any group rendering blocks, (that is, after the last balancing call to [pop\\_group\(\)](#) or [pop\\_group\\_to\\_source\(\)](#)).

## Exceptions

--	--

## Since

1.2

#### 8.2.5.34 get\_group\_target() [2/2]

```
RefPtr< const Surface > Cairo::Context::get_group_target () const
```

Same as the non-const version but returns a reference to a const [Surface](#).

Since

1.2

#### 8.2.5.35 get\_line\_cap()

```
LineCap Cairo::Context::get_line_cap () const
```

Gets the current line cap style, as set by [set\\_line\\_cap\(\)](#)

#### 8.2.5.36 get\_line\_join()

```
LineJoin Cairo::Context::get_line_join () const
```

Gets the current line join style, as set by [set\\_line\\_join\(\)](#)

#### 8.2.5.37 get\_line\_width()

```
double Cairo::Context::get_line_width () const
```

Gets the current line width, as set by [set\\_line\\_width\(\)](#).

Note that the value is unchanged even if the CTM has changed between the calls to [set\\_line\\_width\(\)](#) and [get\\_line\\_width\(\)](#).

#### 8.2.5.38 get\_matrix() [1/2]

```
Matrix Cairo::Context::get_matrix () const
```

Returns the current transformation matrix (CTM)

Since

1.8

#### 8.2.5.39 get\_matrix() [2/2]

```
void Cairo::Context::get_matrix (
    Matrix & matrix)
```

Stores the current transformation matrix (CTM) into matrix.

## Parameters

<i>matrix</i>	return value for the matrix
---------------	-----------------------------

**8.2.5.40 get\_miter\_limit()**

```
double Cairo::Context::get_miter_limit () const
```

Gets the current miter limit, as set by [set\\_miter\\_limit\(\)](#)

**8.2.5.41 get\_operator()**

```
Operator Cairo::Context::get_operator () const
```

Gets the current compositing operator for a cairo [Context](#).

**8.2.5.42 get\_path\_extents()**

```
void Cairo::Context::get_path_extents (
    double & x1,
    double & y1,
    double & x2,
    double & y2) const
```

Computes a bounding box in user-space coordinates covering the points on the current path.

If the current path is empty, returns an empty rectangle ((0,0), (0,0)). Stroke parameters, fill rule, surface dimensions and clipping are not taken into account.

Contrast with [fill\\_extents\(\)](#) and [stroke\\_extents\(\)](#) which return the extents of only the area that would be "inked" by the corresponding drawing operations.

The result of [path\\_extents\(\)](#) is defined as equivalent to the limit of [stroke\\_extents\(\)](#) with [LineCap::ROUND](#) as the line width approaches 0.0, (but never reaching the empty-rectangle returned by [stroke\\_extents\(\)](#) for a line width of 0.0).

Specifically, this means that zero-area sub-paths such as [move\\_to\(\);line\\_to\(\)](#) segments, (even degenerate cases where the coordinates to both calls are identical), will be considered as contributing to the extents. However, a lone [move\\_to\(\)](#) will not contribute to the results of [path\\_extents\(\)](#).

## Parameters

<i>x1</i>	left of the resulting extents
<i>y1</i>	top of the resulting extents
<i>x2</i>	right of the resulting extents
<i>y2</i>	bottom of the resulting extents

## Since

1.6

**8.2.5.43 get\_scaled\_font()**

```
RefPtr< ScaledFont > Cairo::Context::get_scaled_font ()
```

Gets the current scaled font.

Since

1.8

**8.2.5.44 get\_source() [1/2]**

```
RefPtr< Pattern > Cairo::Context::get_source ()
```

Gets the current source pattern for the Context.

**8.2.5.45 get\_source() [2/2]**

```
RefPtr< const Pattern > Cairo::Context::get_source () const
```

**8.2.5.46 get\_source\_for\_surface() [1/2]**

```
RefPtr< SurfacePattern > Cairo::Context::get_source_for_surface ()
```

Gets the current source surface pattern for the Context, if any.

Returns

The source pattern, if it is a surface pattern, else an empty RefPtr.

**8.2.5.47 get\_source\_for\_surface() [2/2]**

```
RefPtr< const SurfacePattern > Cairo::Context::get_source_for_surface () const
```

**8.2.5.48 get\_stroke\_extents()**

```
void Cairo::Context::get_stroke_extents (
    double & x1,
    double & y1,
    double & x2,
    double & y2) const
```

Computes a bounding box in user coordinates covering the area that would be affected, (the "inked" area), by a [stroke\(\)](#) operation given the current path and stroke parameters.

If the current path is empty, returns an empty rectangle ((0,0), (0,0)). [Surface](#) dimensions and clipping are not taken into account.

Note that if the line width is set to exactly zero, then `stroke_extents()` will return an empty rectangle. Contrast with `path_extents()` which can be used to compute the non-empty bounds as the line width approaches zero.

Note that `stroke_extents()` must necessarily do more work to compute the precise inked areas in light of the stroke parameters, so `path_extents()` may be more desirable for sake of performance if non-inked path extents are desired.

## Parameters

<i>x1</i>	left of the resulting extents
<i>y1</i>	top of the resulting extents
<i>x2</i>	right of the resulting extents
<i>y2</i>	bottom of the resulting extents

## See also

[stroke\(\)](#)  
[set\\_line\\_width\(\)](#)  
[set\\_line\\_join\(\)](#)  
[set\\_line\\_cap\(\)](#)  
[set\\_dash\(\)](#)  
[stroke\\_preserve\(\)](#)

8.2.5.49 `get_target()` [1/2]

```
RefPtr< Surface > Cairo::Context::get_target ()
```

Gets the target surface associated with this [Context](#).

## Exceptions

--	--

8.2.5.50 `get_target()` [2/2]

```
RefPtr< const Surface > Cairo::Context::get_target () const
```

8.2.5.51 `get_text_extents()`

```
void Cairo::Context::get_text_extents (
    const std::string & utf8,
    TextExtents & extents) const
```

Gets the extents for a string of text.

The extents describe a user-space rectangle that encloses the "inked" portion of the text, (as it would be drawn by [show\\_text\(\)](#)). Additionally, the `x_advance` and `y_advance` values indicate the amount by which the current point would be advanced by [show\\_text\(\)](#).

Note that whitespace characters do not directly contribute to the size of the rectangle (`extents.width` and `extents.height`). They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the `x_advance` and `y_advance` values.



## Parameters

<i>utf8</i>	a string of text encoded in UTF-8
<i>extents</i>	a TextExtents object

**8.2.5.52 get\_tolerance()**

```
double Cairo::Context::get_tolerance () const
```

Gets the current tolerance value, as set by [set\\_tolerance\(\)](#)

**8.2.5.53 glyph\_path()**

```
void Cairo::Context::glyph_path (
    const std::vector< Glyph > & glyphs)
```

Adds closed paths for the glyphs to the current path.

The generated path if filled, achieves an effect similar to that of [show\\_glyphs\(\)](#).

## Parameters

<i>glyphs</i>	a vector of glyphs
---------------	--------------------

**8.2.5.54 has\_current\_point()**

```
bool Cairo::Context::has_current_point () const
```

Checks if there is a current point defined.

See [get\\_current\\_point\(\)](#) for details on the current point.

## Returns

`true` if a current point is defined.

## Since

1.6

**8.2.5.55 in\_clip()**

```
bool Cairo::Context::in_clip (
    double x,
    double y) const
```

Tests whether the given point is inside the area that would be visible through the current clip, i.e.

the area that would be filled by a [paint\(\)](#) operation.

Return value: A non-zero value if the point is inside, or zero if outside.

**Parameters**

<i>x</i>	X coordinate of the point to test
<i>y</i>	Y coordinate of the point to test

**See also**[clip\(\)](#)[clip\\_preserve\(\)](#)**Since**

1.10

**8.2.5.56 in\_fill()**

```
bool Cairo::Context::in_fill (  
    double x,  
    double y) const
```

Tests whether the given point is inside the area that would be affected by a [fill\(\)](#) operation given the current path and filling parameters.

[Surface](#) dimensions and clipping are not taken into account.

**Parameters**

<i>x</i>	X coordinate of the point to test
<i>y</i>	Y coordinate of the point to test

**Returns**

A non-zero value if the point is inside, or zero if outside.

**See also**[fill\(\)](#)[set\\_fill\\_rule\(\)](#)[fill\\_preserve\(\)](#)**8.2.5.57 in\_stroke()**

```
bool Cairo::Context::in_stroke (  
    double x,  
    double y) const
```

Tests whether the given point is inside the area that would be affected by a [stroke\(\)](#) operation given the current path and stroking parameters.

[Surface](#) dimensions and clipping are not taken into account.

## Parameters

<i>x</i>	X coordinate of the point to test
<i>y</i>	Y coordinate of the point to test

## Returns

A non-zero value if the point is inside, or zero if outside.

## See also

[stroke\(\)](#)  
[set\\_line\\_width\(\)](#)  
[set\\_line\\_join\(\)](#)  
[set\\_line\\_cap\(\)](#)  
[set\\_dash\(\)](#)  
[stroke\\_preserve\(\)](#).

**8.2.5.58 line\_to()**

```
void Cairo::Context::line_to (
    double x,
    double y)
```

Adds a line to the path from the current point to position (x, y) in user-space coordinates.

After this call the current point will be (x, y).

If there is no current point before the call to [line\\_to\(\)](#) this function will behave as [move\\_to\(x, y\)](#).

## Parameters

<i>x</i>	the X coordinate of the end of the new line
<i>y</i>	the Y coordinate of the end of the new line

**8.2.5.59 mask() [1/2]**

```
void Cairo::Context::mask (
    const RefPtr< const Pattern > & pattern)
```

A drawing operator that paints the current source using the alpha channel of pattern as a mask.

(Opaque areas of mask are painted with the source, transparent areas are not painted.)

## Parameters

<i>pattern</i>	a <a href="#">Pattern</a>
----------------	---------------------------

**8.2.5.60 mask() [2/2]**

```
void Cairo::Context::mask (
    const RefPtr< const Surface > & surface,
    double surface_x,
    double surface_y)
```

A drawing operator that paints the current source using the alpha channel of surface as a mask.

(Opaque areas of surface are painted with the source, transparent areas are not painted.)

## Parameters

<i>surface</i>	a <a href="#">Surface</a>
<i>surface</i> ↔ <i>_x</i>	X coordinate at which to place the origin of surface
<i>surface</i> ↔ <i>_y</i>	Y coordinate at which to place the origin of surface

**8.2.5.61 move\_to()**

```
void Cairo::Context::move_to (
    double x,
    double y)
```

If the current subpath is not empty, begin a new subpath.

After this call the current point will be (x, y).

## Parameters

<i>x</i>	the X coordinate of the new position
<i>y</i>	the Y coordinate of the new position

**8.2.5.62 operator=()**

```
Context & Cairo::Context::operator= (
    const Context & ) [delete]
```

**8.2.5.63 paint()**

```
void Cairo::Context::paint ()
```

A drawing operator that paints the current source everywhere within the current clip region.

**8.2.5.64 paint\_with\_alpha()**

```
void Cairo::Context::paint_with_alpha (
    double alpha)
```

A drawing operator that paints the current source everywhere within the current clip region using a mask of constant alpha value alpha.

The effect is similar to [paint\(\)](#), but the drawing is faded out using the alpha value.

## Parameters

<i>alpha</i>	an alpha value, between 0 (transparent) and 1 (opaque)
--------------	--------------------------------------------------------

### 8.2.5.65 pop\_group()

```
RefPtr< Pattern > Cairo::Context::pop_group ()
```

Terminates the redirection begun by a call to [push\\_group\(\)](#) or [push\\_group\\_with\\_content\(\)](#) and returns a new pattern containing the results of all drawing operations performed to the group.

The [pop\\_group\(\)](#) function calls [restore\(\)](#), (balancing a call to [save\(\)](#) by the [push\\_group](#) function), so that any changes to the graphics state will not be visible outside the group.

#### Returns

a (surface) pattern containing the results of all drawing operations performed to the group.

#### Since

1.2

### 8.2.5.66 pop\_group\_to\_source()

```
void Cairo::Context::pop_group_to_source ()
```

Terminates the redirection begun by a call to [push\\_group\(\)](#) or [push\\_group\\_with\\_content\(\)](#) and installs the resulting pattern as the source pattern in the given cairo [Context](#).

The behavior of this function is equivalent to the sequence of operations:

```
RefPtr<Pattern> group = cr->pop_group();  
cr->set_source(group);
```

but is more convenient as there is no need for a variable to store the short-lived pointer to the pattern.

The [pop\\_group\(\)](#) function calls [restore\(\)](#), (balancing a call to [save\(\)](#) by the [push\\_group](#) function), so that any changes to the graphics state will not be visible outside the group.

#### Since

1.2

### 8.2.5.67 push\_group()

```
void Cairo::Context::push_group ()
```

Temporarily redirects drawing to an intermediate surface known as a group.

The redirection lasts until the group is completed by a call to [pop\\_group\(\)](#) or [pop\\_group\\_to\\_source\(\)](#). These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

This group functionality can be convenient for performing intermediate compositing. One common use of a group is to render objects as opaque within the group, (so that they occlude each other), and then blend the result with translucence onto the destination.

Groups can be nested arbitrarily deep by making balanced calls to [push\\_group\(\)/pop\\_group\(\)](#). Each call pushes/pops the new target group onto/from a stack.

The [push\\_group\(\)](#) function calls [save\(\)](#) so that any changes to the graphics state will not be visible outside the group, (the [pop\\_group](#) functions call [restore\(\)](#)).

By default the intermediate group will have a content type of `CONTENT_COLOR_ALPHA`. Other content types can be chosen for the group by using [push\\_group\\_with\\_content\(\)](#) instead.

As an example, here is how one might fill and stroke a path with translucence, but without any portion of the fill being visible under the stroke:

```
cr->push_group();  
cr->set_source(fill_pattern);  
cr->fill_preserve();  
cr->set_source(stroke_pattern);  
cr->stroke();  
cr->pop_group_to_source();  
cr->paint_with_alpha(alpha);
```

Since

1.2

### 8.2.5.68 push\_group\_with\_content()

```
void Cairo::Context::push_group_with_content (
    Content content)
```

Temporarily redirects drawing to an intermediate surface known as a group.

The redirection lasts until the group is completed by a call to [pop\\_group\(\)](#) or [pop\\_group\\_to\\_source\(\)](#). These calls provide the result of any drawing to the group as a pattern, (either as an explicit object, or set as the source pattern).

The group will have a content type of @content. The ability to control this content type is the only distinction between this function and [push\\_group\(\)](#) which you should see for a more detailed description of group rendering.

#### Parameters

<i>content</i>	indicates the type of group that will be created
----------------	--------------------------------------------------

Since

1.2

### 8.2.5.69 rectangle()

```
void Cairo::Context::rectangle (
    double x,
    double y,
    double width,
    double height)
```

Adds a closed-subpath rectangle of the given size to the current path at position (x, y) in user-space coordinates.

This function is logically equivalent to:

```
context->move_to(x, y);
context->rel_line_to(width, 0);
context->rel_line_to(0, height);
context->rel_line_to(-width, 0);
context->close_path();
```

#### Parameters

<i>x</i>	the X coordinate of the top left corner of the rectangle
<i>y</i>	the Y coordinate to the top left corner of the rectangle
<i>width</i>	the width of the rectangle
<i>height</i>	the height of the rectangle

### 8.2.5.70 rel\_curve\_to()

```
void Cairo::Context::rel_curve_to (
    double dx1,
    double dy1,
    double dx2,
    double dy2,
    double dx3,
    double dy3)
```

Relative-coordinate version of [curve\\_to\(\)](#).

All offsets are relative to the current point. Adds a cubic Bezier spline to the path from the current point to a point offset from the current point by (dx3, dy3), using points offset by (dx1, dy1) and (dx2, dy2) as the control points. After this call the current point will be offset by (dx3, dy3).

Given a current point of (x, y),  
[rel\\_curve\\_to](#)(dx1, dy1, dx2, dy2, dx3, dy3)

is logically equivalent to  
[curve\\_to](#)(x + dx1, y + dy1, x + dx2, y + dy2, x + dx3, y + dy3).

#### Parameters

<i>dx1</i>	the X offset to the first control point
<i>dy1</i>	the Y offset to the first control point
<i>dx2</i>	the X offset to the second control point
<i>dy2</i>	the Y offset to the second control point
<i>dx3</i>	the X offset to the end of the curve
<i>dy3</i>	the Y offset to the end of the curve

It is an error to call this function with no current point. Doing so will cause this to shutdown with a status of CAIRO↵\_STATUS\_NO\_CURRENT\_POINT. Caiomm will then throw an exception.

### 8.2.5.71 rel\_line\_to()

```
void Cairo::Context::rel_line_to (
    double dx,
    double dy)
```

Relative-coordinate version of [line\\_to\(\)](#).

Adds a line to the path from the current point to a point that is offset from the current point by (dx, dy) in user space. After this call the current point will be offset by (dx, dy).

Given a current point of (x, y),  
[rel\\_line\\_to](#)(dx, dy)

is logically equivalent to  
[line\\_to](#)(x + dx, y + dy).

#### Parameters

<i>dx</i>	the X offset to the end of the new line
<i>dy</i>	the Y offset to the end of the new line

It is an error to call this function with no current point. Doing so will cause this to shutdown with a status of CAIRO↵\_STATUS\_NO\_CURRENT\_POINT. Caiomm will then throw an exception.

### 8.2.5.72 rel\_move\_to()

```
void Cairo::Context::rel_move_to (
    double dx,
    double dy)
```

If the current subpath is not empty, begin a new subpath.

After this call the current point will offset by (x, y).

Given a current point of (x, y),  
[rel\\_move\\_to](#)(dx, dy)

is logically equivalent to  
[move\\_to](#)(x + dx, y + dy)

#### Parameters

<i>dx</i>	the X offset
<i>dy</i>	the Y offset

It is an error to call this function with no current point. Doing so will cause this to shutdown with a status of CAIRO↵\_STATUS\_NO\_CURRENT\_POINT. Cairomm will then throw an exception.

### 8.2.5.73 reset\_clip()

```
void Cairo::Context::reset_clip ()
```

Reset the current clip region to its original, unrestricted state.

That is, set the clip region to an infinitely large shape containing the target surface. Equivalently, if infinity is too hard to grasp, one can imagine the clip region being reset to the exact bounds of the target surface.

Note that code meant to be reusable should not call [reset\\_clip\(\)](#) as it will cause results unexpected by higher-level code which calls [clip\(\)](#). Consider using [save\(\)](#) and [restore\(\)](#) around [clip\(\)](#) as a more robust means of temporarily restricting the clip region.

### 8.2.5.74 restore()

```
void Cairo::Context::restore ()
```

Restores cr to the state saved by a preceding call to [save\(\)](#) and removes that state from the stack of saved states.

#### See also

[save\(\)](#), [SaveGuard](#)

### 8.2.5.75 rotate()

```
void Cairo::Context::rotate (
    double angle_radians)
```

Modifies the current transformation matrix (CTM) by rotating the user-space axes by angle radians.

The rotation of the axes takes places after any existing transformation of user space. The rotation direction for positive angles is from the positive X axis toward the positive Y axis.



## Parameters

<i>angle</i>	angle (in radians) by which the user-space axes will be rotated
--------------	-----------------------------------------------------------------

**8.2.5.76 rotate\_degrees()**

```
void Cairo::Context::rotate_degrees (
    double angle_degrees)
```

A convenience wrapper around [rotate\(\)](#) that accepts angles in degrees.

## Parameters

<i>angle_degrees</i>	angle (in degrees) by which the user-space axes should be rotated
----------------------	-------------------------------------------------------------------

**8.2.5.77 save()**

```
void Cairo::Context::save ()
```

Makes a copy of the current state of the [Context](#) and saves it on an internal stack of saved states.

When [restore\(\)](#) is called, it will be restored to the saved state. Multiple calls to [save\(\)](#) and [restore\(\)](#) can be nested; each call to [restore\(\)](#) restores the state from the matching paired [save\(\)](#).

It isn't necessary to clear all saved states before a `cairo_t` is freed. Any saved states will be freed when the [Context](#) is destroyed.

## See also

[restore\(\)](#), [SaveGuard](#)

**8.2.5.78 scale()**

```
void Cairo::Context::scale (
    double sx,
    double sy)
```

Modifies the current transformation matrix (CTM) by scaling the X and Y user-space axes by `sx` and `sy` respectively.

The scaling of the axes takes place after any existing transformation of user space.

## Parameters

<i>sx</i>	scale factor for the X dimension
<i>sy</i>	scale factor for the Y dimension

### 8.2.5.79 select\_font\_face()

```
void Cairo::Context::select_font_face (
    const std::string & family,
    ToyFontFace::Slant slant,
    ToyFontFace::Weight weight)
```

Selects a family and style of font from a simplified description as a family name, slant and weight.

[Cairo](#) provides no operation to list available family names on the system (this is a "toy", remember), but the standard CSS2 generic family names, ("serif", "sans-serif", "cursive", "fantasy", "monospace"), are likely to work as expected.

Note: The [select\\_font\\_face\(\)](#) function call is part of what the cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications.

If *family* starts with the string "@cairo:", or if no native font backends are compiled in, cairo will use an internal font family. The internal font family recognizes many modifiers in the @family string, most notably, it recognizes the string "monospace". That is, the family name "@cairo:monospace" will use the monospace version of the internal font family.

For "real" font selection, see the font-backend-specific [Cairo::FontFace::create](#) functions for the font backend you are using. (For example, if you are using the freetype-based [cairo-ft](#) font backend, see [Cairo::FtFontFace::create\(\)](#).) The resulting font face could then be used with [Cairo::ScaledFont::create\(\)](#) and [set\\_scaled\\_font\(\)](#).

Similarly, when using the "real" font support, you can call directly into the underlying font system, (such as fontconfig or freetype), for operations such as listing available fonts, etc.

It is expected that most applications will need to use a more comprehensive font handling and text layout library, (for example, pango), in conjunction with cairo.

If text is drawn without a call to [select\\_font\\_face\(\)](#), (nor [set\\_font\\_face\(\)](#) nor [set\\_scaled\\_font\(\)](#)), the default family is platform-specific, but is essentially "sans-serif". Default slant is [Cairo::FONT\\_SLANT\\_NORMAL](#), and default weight is [Cairo::FONT\\_WEIGHT\\_NORMAL](#).

This function is equivalent to a call to [Cairo::ToyFontFace::create\(\)](#) followed by [set\\_font\\_face\(\)](#).

#### Parameters

<i>family</i>	a font family name, encoded in UTF-8
<i>slant</i>	the slant for the font
<i>weight</i>	the weight for the font

### 8.2.5.80 set\_antialias()

```
void Cairo::Context::set_antialias (
    Antialias antialias)
```

Set the antialiasing mode of the rasterizer used for drawing shapes.

This value is a hint, and a particular backend may or may not support a particular value. At the current time, no backend supports [Cairo::ANTIALIAS\\_SUBPIXEL](#) when drawing shapes.

Note that this option does not affect text rendering, instead see [FontOptions::set\\_antialias\(\)](#).

## Parameters

<i>antialias</i>	the new antialiasing mode
------------------	---------------------------

**8.2.5.81 set\_dash()** [1/2]

```
void Cairo::Context::set_dash (
    const std::valarray< double > & dashes,
    double offset)
```

Alternate version of [set\\_dash\(\)](#).

You'll probably want to use the one that takes a **std::vector** argument instead.

**8.2.5.82 set\_dash()** [2/2]

```
void Cairo::Context::set_dash (
    const std::vector< double > & dashes,
    double offset)
```

Sets the dash pattern to be used by [stroke\(\)](#).

A dash pattern is specified by dashes, an array of positive values. Each value provides the user-space length of alternate "on" and "off" portions of the stroke. The offset specifies an offset into the pattern at which the stroke begins.

Each "on" segment will have caps applied as if the segment were a separate sub-path. In particular, it is valid to use an "on" length of 0.0 with Cairo::LineCap::ROUND or Cairo::LineCap::SQUARE in order to distributed dots or squares along a path.

Note: The length values are in user-space units as evaluated at the time of stroking. This is not necessarily the same as the user space at the time of [set\\_dash\(\)](#).

If dashes is empty dashing is disabled. If the size of dashes is 1, a symmetric pattern is assumed with alternating on and off portions of the size specified by the single value in dashes.

It is invalid for any value in dashes to be negative, or for all values to be 0. If this is the case, an exception will be thrown

## Parameters

<i>dashes</i>	an array specifying alternate lengths of on and off portions
<i>offset</i>	an offset into the dash pattern at which the stroke should start

## Exceptions

--	--

**8.2.5.83 set\_fill\_rule()**

```
void Cairo::Context::set_fill_rule (
    FillRule fill_rule)
```

Set the current fill rule within the cairo [Context](#).

The fill rule is used to determine which regions are inside or outside a complex (potentially self-intersecting) path. The current fill rule affects both [fill\(\)](#) and [clip\(\)](#). See FillRule for details on the semantics of each available fill rule.

The default fill rule is Cairo::FILL\_RULE\_WINDING.

## Parameters

<i>fill_rule</i>	a fill rule, specified as a FillRule
------------------	--------------------------------------

**8.2.5.84 set\_font\_face()**

```
void Cairo::Context::set_font_face (
    const RefPtr< const FontFace > & font_face)
```

Replaces the current font face in the context with *font\_face*.

The replaced font face in the context will be destroyed if there are no other references to it.

## Parameters

<i>font_face</i>	a font face
------------------	-------------

**8.2.5.85 set\_font\_matrix()**

```
void Cairo::Context::set_font_matrix (
    const Matrix & matrix)
```

Sets the current font matrix to @matrix.

The font matrix gives a transformation from the design space of the font (in this space, the em-square is 1 unit by 1 unit) to user space. Normally, a simple scale is used (see [set\\_font\\_size\(\)](#)), but a more complex font matrix can be used to shear the font or stretch it unequally along the two axes

## Parameters

<i>matrix</i>	a <a href="#">Cairo::Matrix</a> describing a transform to be applied to the current font.
---------------	-------------------------------------------------------------------------------------------

**8.2.5.86 set\_font\_options()**

```
void Cairo::Context::set_font_options (
    const FontOptions & options)
```

Sets a set of custom font rendering options.

Rendering options are derived by merging these options with the options derived from underlying surface; if the value in *options* has a default value (like [Cairo::ANTIALIAS\\_DEFAULT](#)), then the value from the surface is used.

## Parameters

<i>options</i>	font options to use
----------------	---------------------

### 8.2.5.87 set\_font\_size()

```
void Cairo::Context::set_font_size (
    double size)
```

Sets the current font matrix to a scale by a factor of *size*, replacing any font matrix previously set with [set\\_font\\_size\(\)](#) or [set\\_font\\_matrix\(\)](#).

This results in a font size of *size* user space units. (More precisely, this matrix will result in the font's em-square being a @size by *size* square in user space.)

If text is drawn without a call to [set\\_font\\_size\(\)](#), (nor [set\\_font\\_matrix\(\)](#) nor [set\\_scaled\\_font\(\)](#)), the default font size is 10.0.

## Parameters

<i>size</i>	the new font size, in user space units)
-------------	-----------------------------------------

**8.2.5.88 set\_identity\_matrix()**

```
void Cairo::Context::set_identity_matrix ()
```

Resets the current transformation matrix (CTM) by setting it equal to the identity matrix.

That is, the user-space and device-space axes will be aligned and one user-space unit will transform to one device-space unit.

**8.2.5.89 set\_line\_cap()**

```
void Cairo::Context::set_line_cap (  
    LineCap line_cap)
```

Sets the current line cap style within the cairo [Context](#).

See LineCap for details about how the available line cap styles are drawn.

As with the other stroke parameters, the current line cap style is examined by [stroke\(\)](#), [stroke\\_extents\(\)](#), and [stroke\\_to\\_path\(\)](#), but does not have any effect during path construction.

The default line cap style is Cairo::LineCap::BUTT.

## Parameters

<i>line_cap</i>	a line cap style, as a LineCap
-----------------	--------------------------------

**8.2.5.90 set\_line\_join()**

```
void Cairo::Context::set_line_join (  
    LineJoin line_join)
```

Sets the current line join style within the cairo [Context](#).

See LineJoin for details about how the available line join styles are drawn.

As with the other stroke parameters, the current line join style is examined by [stroke\(\)](#), [stroke\\_extents\(\)](#), and [stroke\\_to\\_path\(\)](#), but does not have any effect during path construction.

The default line join style is Cairo::LineJoin::MITER.

## Parameters

<i>line_join</i>	a line joint style, as a LineJoin
------------------	-----------------------------------

### 8.2.5.91 set\_line\_width()

```
void Cairo::Context::set_line_width (
    double width)
```

Sets the current line width within the cairo [Context](#).

The line width specifies the diameter of a pen that is circular in user-space, (though device-space pen may be an ellipse in general due to scaling/shear/rotation of the CTM).

Note: When the description above refers to user space and CTM it refers to the user space and CTM in effect at the time of the stroking operation, not the user space and CTM in effect at the time of the call to [set\\_line\\_width\(\)](#). The simplest usage makes both of these spaces identical. That is, if there is no change to the CTM between a call to [set\\_line\\_width\(\)](#) and the stroking operation, then one can just pass user-space values to [set\\_line\\_width\(\)](#) and ignore this note.

As with the other stroke parameters, the current line cap style is examined by [stroke\(\)](#), [stroke\\_extents\(\)](#), and [stroke\\_to\\_path\(\)](#), but does not have any effect during path construction.

The default line width value is 2.0.

#### Parameters

<i>width</i>	a line width, as a user-space value
--------------	-------------------------------------

### 8.2.5.92 set\_matrix()

```
void Cairo::Context::set_matrix (
    const Matrix & matrix)
```

Modifies the current transformation matrix (CTM) by setting it equal to matrix.

#### Parameters

<i>matrix</i>	a transformation matrix from user space to device space
---------------	---------------------------------------------------------

### 8.2.5.93 set\_miter\_limit()

```
void Cairo::Context::set_miter_limit (
    double limit)
```

Sets the current miter limit within the cairo context.

If the current line join style is set to Cairo::LineJoin::MITER (see [set\\_line\\_join\(\)](#)), the miter limit is used to determine whether the lines should be joined with a bevel instead of a miter. Cairo divides the length of the miter by the line width. If the result is greater than the miter limit, the style is converted to a bevel.

As with the other stroke parameters, the current line miter limit is examined by [stroke\(\)](#), [stroke\\_extents\(\)](#), and [stroke\\_to\\_path\(\)](#), but does not have any effect during path construction.

The default miter limit value is 10.0, which will convert joins with interior angles less than 11 degrees to bevels instead of miters. For reference, a miter limit of 2.0 makes the miter cutoff at 60 degrees, and a miter limit of 1.414 makes the cutoff at 90 degrees.

A miter limit for a desired angle can be computed as:  $\text{miter\_limit} = 1/\sin(\text{angle}/2)$

## Parameters

<i>limit</i>	miter limit to set
--------------	--------------------

**8.2.5.94 set\_operator()**

```
void Cairo::Context::set_operator (
    Operator op)
```

Sets the compositing operator to be used for all drawing operations.

See [Operator](#) for details on the semantics of each available compositing operator.

## Parameters

<i>op</i>	a compositing operator, specified as a <a href="#">Operator</a>
-----------	-----------------------------------------------------------------

**8.2.5.95 set\_scaled\_font()**

```
void Cairo::Context::set_scaled_font (
    const RefPtr< const ScaledFont > & scaled_font)
```

Replaces the current font face, font matrix, and font options in the context with those of the *scaled\_font*.

Except for some translation, the current CTM of the context should be the same as that of the `#cairo_scaled_font_t`, which can be accessed using [Cairo::ScaledFont::get\\_ctm\(\)](#).

## Parameters

<i>scaled_font</i>	a scaled font
--------------------	---------------

Since

1.8

**8.2.5.96 set\_source() [1/2]**

```
void Cairo::Context::set_source (
    const RefPtr< const Pattern > & source)
```

Sets the source pattern within the [Context](#) to source.

This [Pattern](#) will then be used for any subsequent drawing operation until a new source pattern is set.

Note: The [Pattern](#)'s transformation matrix will be locked to the user space in effect at the time of [set\\_source\(\)](#). This means that further modifications of the current transformation matrix will not affect the source pattern.



## Parameters

<i>source</i>	a <a href="#">Pattern</a> to be used as the source for subsequent drawing operations.
---------------	---------------------------------------------------------------------------------------

## See also

[Pattern::set\\_matrix\(\)](#)  
[set\\_source\\_rgb\(\)](#)  
[set\\_source\\_rgba\(\)](#)  
[set\\_source\(const RefPtr<Surface>& surface, double x, double y\)](#)

**8.2.5.97 set\_source()** [2/2]

```
void Cairo::Context::set_source (
    const RefPtr< Surface > & surface,
    double x,
    double y)
```

This is a convenience function for creating a pattern from a [Surface](#) and setting it as the source.

The x and y parameters give the user-space coordinate at which the [Surface](#) origin should appear. (The [Surface](#) origin is its upper-left corner before any transformation has been applied.) The x and y patterns are negated and then set as translation values in the pattern matrix.

Other than the initial translation pattern matrix, as described above, all other pattern attributes, (such as its extend mode), are set to the default values as in [Context::create\(const RefPtr<Surface>& target\)](#). The resulting pattern can be queried with [get\\_source\(\)](#) so that these attributes can be modified if desired, (eg. to create a repeating pattern with [Pattern::set\\_extend\(\)](#)).

## Parameters

<i>surface</i>	a <a href="#">Surface</a> to be used to set the source pattern
<i>x</i>	User-space X coordinate for surface origin
<i>y</i>	User-space Y coordinate for surface origin

**8.2.5.98 set\_source\_rgb()**

```
void Cairo::Context::set_source_rgb (
    double red,
    double green,
    double blue)
```

Sets the source pattern within the [Context](#) to an opaque color.

This opaque color will then be used for any subsequent drawing operation until a new source pattern is set.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

## Parameters

<i>red</i>	red component of color
<i>green</i>	green component of color
<i>blue</i>	blue component of color

## See also

[set\\_source\\_rgba\(\)](#)[set\\_source\(\)](#)**8.2.5.99 set\_source\_rgba()**

```
void Cairo::Context::set_source_rgba (
    double red,
    double green,
    double blue,
    double alpha)
```

Sets the source pattern within the [Context](#) to a translucent color.

This color will then be used for any subsequent drawing operation until a new source pattern is set.

The color and alpha components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

## Parameters

<i>red</i>	red component of color
<i>green</i>	green component of color
<i>blue</i>	blue component of color
<i>alpha</i>	alpha component of color

## See also

[set\\_source\\_rgb\(\)](#)[set\\_source\(\)](#)**8.2.5.100 set\_tolerance()**

```
void Cairo::Context::set_tolerance (
    double tolerance)
```

Sets the tolerance used when converting paths into trapezoids.

Curved segments of the path will be subdivided until the maximum deviation between the original path and the polygonal approximation is less than tolerance. The default value is 0.1. A larger value will give better performance, a smaller value, better appearance. (Reducing the value from the default value of 0.1 is unlikely to improve appearance significantly.) The accuracy of paths within [Cairo](#) is limited by the precision of its internal arithmetic, and the prescribed @tolerance is restricted to the smallest representable internal value.

## Parameters

<i>tolerance</i>	the tolerance, in device units (typically pixels)
------------------	---------------------------------------------------

**8.2.5.101 show\_glyphs()**

```
void Cairo::Context::show_glyphs (
    const std::vector< Glyph > & glyphs)
```

A drawing operator that generates the shape from an array of glyphs, rendered according to the current font face, font size (font matrix), and font options.

## Parameters

<i>glyphs</i>	vector of glyphs to show
<i>num_glyphs</i>	number of glyphs to show

**8.2.5.102 show\_page()**

```
void Cairo::Context::show_page ()
```

Emits and clears the current page for backends that support multiple pages.

Use [copy\\_page\(\)](#) if you don't want to clear the page.

This is a convenience function that simply calls [Surface::show\\_page\(\)](#) on *cr*'s target.

**8.2.5.103 show\_text()**

```
void Cairo::Context::show_text (
    const std::string & utf8)
```

A drawing operator that generates the shape from a string of UTF-8 characters, rendered according to the current font\_face, font\_size (font\_matrix), and font\_options.

This function first computes a set of glyphs for the string of text. The first glyph is placed so that its origin is at the current point. The origin of each subsequent glyph is offset from that of the previous glyph by the advance values of the previous glyph.

After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for easy display of a single logical string with multiple calls to [show\\_text\(\)](#).

Note: The [show\\_text\(\)](#) function call is part of what the cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See [show\\_glyphs\(\)](#) for the "real" text display API in cairo.

## Parameters

<i>utf8</i>	a string containing text encoded in UTF-8
-------------	-------------------------------------------

### 8.2.5.104 show\_text\_glyphs()

```
void Cairo::Context::show_text_glyphs (
    const std::string & utf8,
    const std::vector< Glyph > & glyphs,
    const std::vector< TextCluster > & clusters,
    TextClusterFlags cluster_flags)
```

This operation has rendering effects similar to [show\\_glyphs\(\)](#) but, if the target surface supports it, uses the provided text and cluster mapping to embed the text for the glyphs shown in the output.

If the target does not support the extended attributes, this function acts like the basic [show\\_glyphs\(\)](#) as if it had been passed *glyphs* and *num\_glyphs*.

The mapping between *utf8* and *glyphs* is provided by an array of [clusters](#). Each cluster covers a number of text bytes and glyphs, and neighboring clusters cover neighboring areas of *utf8* and *glyphs*. The clusters should collectively cover *utf8* and *glyphs* in entirety.

The first cluster always covers bytes from the beginning of *utf8*. If *cluster\_flags* do not have the [Cairo::TEXT\\_CLUSTER\\_FLAG\\_BACKWARD](#) set, the first cluster also covers the beginning of *glyphs*, otherwise it covers the end of the *glyphs* array and following clusters move backward.

See [Cairo::TextCluster](#) for constraints on valid clusters.

#### Parameters

<i>utf8</i>	a string of text encoded in UTF-8
<i>glyphs</i>	vector of glyphs to show
<i>clusters</i>	vector of cluster mapping information
<i>cluster_flags</i>	cluster mapping flags

#### Since

1.8

### 8.2.5.105 stroke()

```
void Cairo::Context::stroke ()
```

A drawing operator that strokes the current [Path](#) according to the current line width, line join, line cap, and dash settings.

After [stroke\(\)](#), the current [Path](#) will be cleared from the cairo [Context](#).

See also

[set\\_line\\_width\(\)](#)  
[set\\_line\\_join\(\)](#)  
[set\\_line\\_cap\(\)](#)  
[set\\_dash\(\)](#)  
[stroke\\_preserve\(\)](#).

Note: Degenerate segments and sub-paths are treated specially and provide a useful result. These can result in two different situations:

1. Zero-length "on" segments set in [set\\_dash\(\)](#). If the cap style is Cairo::LineCap::ROUND or Cairo::LineCap::SQUARE then these segments will be drawn as circular dots or squares respectively. In the case of Cairo::LineCap::SQUARE, the orientation of the squares is determined by the direction of the underlying path.
2. A sub-path created by [move\\_to\(\)](#) followed by either a [close\\_path\(\)](#) or one or more calls to [line\\_to\(\)](#) to the same coordinate as the [move\\_to\(\)](#). If the cap style is Cairo::LineCap::ROUND then these sub-paths will be drawn as circular dots. Note that in the case of Cairo::LineCap::SQUARE a degenerate sub-path will not be drawn at all, (since the correct orientation is indeterminate).

In no case will a cap style of Cairo::LineCap::BUTT cause anything to be drawn in the case of either degenerate segments or sub-paths.

#### 8.2.5.106 stroke\_preserve()

```
void Cairo::Context::stroke_preserve ()
```

A drawing operator that strokes the current [Path](#) according to the current line width, line join, line cap, and dash settings.

Unlike [stroke\(\)](#), [stroke\\_preserve\(\)](#) preserves the [Path](#) within the cairo [Context](#).

See also

[set\\_line\\_width\(\)](#)  
[set\\_line\\_join\(\)](#)  
[set\\_line\\_cap\(\)](#)  
[set\\_dash\(\)](#)  
[stroke\\_preserve\(\)](#).

#### 8.2.5.107 text\_path()

```
void Cairo::Context::text_path (
    const std::string & utf8)
```

Adds closed paths for text to the current path.

The generated path if filled, achieves an effect similar to that of [show\\_text\(\)](#).

Text conversion and positioning is done similar to [show\\_text\(\)](#).

Like [show\\_text\(\)](#), After this call the current point is moved to the origin of where the next glyph would be placed in this same progression. That is, the current point will be at the origin of the final glyph offset by its advance values. This allows for chaining multiple calls to [text\\_path\(\)](#) without having to set current point in between.

Note: The [text\\_path\(\)](#) function call is part of what the cairo designers call the "toy" text API. It is convenient for short demos and simple programs, but it is not expected to be adequate for serious text-using applications. See [glyph\\_path\(\)](#) for the "real" text path API in cairo.

## Parameters

<i>utf8</i>	a string of text encoded in UTF-8
-------------	-----------------------------------

**8.2.5.108 transform()**

```
void Cairo::Context::transform (
    const Matrix & matrix)
```

Modifies the current transformation matrix (CTM) by applying matrix as an additional transformation.

The new transformation of user space takes place after any existing transformation.

## Parameters

<i>matrix</i>	a transformation to be applied to the user-space axes
---------------	-------------------------------------------------------

**8.2.5.109 translate()**

```
void Cairo::Context::translate (
    double tx,
    double ty)
```

Modifies the current transformation matrix (CTM) by translating the user-space origin by (tx, ty).

This offset is interpreted as a user-space coordinate according to the CTM in place before the new call to translate. In other words, the translation of the user-space origin takes place after any existing transformation.

## Parameters

<i>tx</i>	amount to translate in the X direction
<i>ty</i>	amount to translate in the Y direction

**8.2.5.110 unset\_dash()**

```
void Cairo::Context::unset_dash ()
```

This function disables a dash pattern that was set with [set\\_dash\(\)](#)

**8.2.5.111 user\_to\_device()**

```
void Cairo::Context::user_to_device (
    double & x,
    double & y) const
```

Transform a coordinate from user space to device space by multiplying the given point by the current transformation matrix (CTM).

## Parameters

<i>x</i>	X value of coordinate (in/out parameter)
<i>y</i>	Y value of coordinate (in/out parameter)

**8.2.5.112 user\_to\_device\_distance()**

```
void Cairo::Context::user_to_device_distance (
    double & dx,
    double & dy) const
```

Transform a distance vector from user space to device space.

This function is similar to [user\\_to\\_device\(\)](#) except that the translation components of the CTM will be ignored when transforming (dx,dy).

## Parameters

<i>dx</i>	X component of a distance vector (in/out parameter)
<i>dy</i>	Y component of a distance vector (in/out parameter)

**8.2.6 Member Data Documentation****8.2.6.1 m\_cobject**

```
cobject* Cairo::Context::m_cobject [protected]
```

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

- cairomm/context.h

**8.3 Cairo::Device Class Reference**

Devices are the abstraction [Cairo](#) employs for the rendering system used by a `cairo_surface_t`.

```
#include <cairomm/device.h>
```

**Classes**

- class [Lock](#)  
A convenience class for acquiring a [Device](#) object in an exception-safe manner.

## Public Types

- enum class [DeviceType](#) {  
[DRM](#) = CAIRO\_DEVICE\_TYPE\_DRM ,  
[GL](#) = CAIRO\_DEVICE\_TYPE\_GL ,  
[SCRIPT](#) = CAIRO\_DEVICE\_TYPE\_SCRIPT ,  
[XCB](#) = CAIRO\_DEVICE\_TYPE\_XCB ,  
[XLIB](#) = CAIRO\_DEVICE\_TYPE\_XLIB ,  
[XML](#) = CAIRO\_DEVICE\_TYPE\_XML }
- typedef cairo\_device\_t [cobject](#)

## Public Member Functions

- [Device](#) (cairo\_device\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- virtual [~Device](#) ()
- [DeviceType](#) [get\\_type](#) () const  
*This function returns the type of the device.*
- void [flush](#) ()  
*Finish any pending operations for the device and also restore any temporary modifications cairo has made to the device's state.*
- void [finish](#) ()  
*This function finishes the device and drops all references to external resources.*
- void [acquire](#) ()  
*Acquires the device for the current thread.*
- void [release](#) ()  
*Releases a device previously acquired using [acquire\(\)](#).*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Protected Attributes

- [cobject](#) \* [m\\_cobject](#)

## 8.3.1 Detailed Description

Devices are the abstraction [Cairo](#) employs for the rendering system used by a [cairo\\_surface\\_t](#).

You can get the device of a surface using [Surface::get\\_device\(\)](#).

Devices are created using custom functions specific to the rendering system you want to use. See the documentation for the surface types for those functions.

An important function that devices fulfill is sharing access to the rendering system between [Cairo](#) and your application. If you want to access a device directly that you used to draw to with [Cairo](#), you must first call [flush\(\)](#) to ensure that [Cairo](#) finishes all operations on the device and resets it to a clean state.

[Cairo](#) also provides the functions [acquire\(\)](#) and [release\(\)](#) to synchronize access to the rendering system in a multi-threaded environment. This is done internally, but can also be used by applications. There is also a [Device::Lock](#) convenience class that allows the device to be acquired and released in an exception-safe manner.

This is a reference-counted object that should be used via [Cairo::RefPtr](#).

Since

1.10



## 8.3.2 Member Typedef Documentation

### 8.3.2.1 cobject

```
typedef cairo_device_t Cairo::Device::cobject
```

## 8.3.3 Member Enumeration Documentation

### 8.3.3.1 DeviceType

```
enum class Cairo::Device::DeviceType [strong]
```

Since

1.10

Enumerator

DRM	
GL	
SCRIPT	
XCB	
XLIB	
XML	

## 8.3.4 Constructor & Destructor Documentation

### 8.3.4.1 Device()

```
Cairo::Device::Device (
    cairo_device_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.3.4.2 ~Device()

```
virtual Cairo::Device::~~Device () [virtual]
```

## 8.3.5 Member Function Documentation

### 8.3.5.1 `acquire()`

```
void Cairo::Device::acquire ()
```

Acquires the device for the current thread.

This function will block until no other thread has acquired the device.

If no exception is thrown, you successfully acquired the device. From now on your thread owns the device and no other thread will be able to acquire it until a matching call to `release()`. It is allowed to recursively acquire the device multiple times from the same thread.

#### Note

It is recommended to use `Device::Lock` to acquire devices in an exception-safe manner, rather than acquiring and releasing the device manually.

#### Warning

You must never acquire two different devices at the same time unless this is explicitly allowed. Otherwise the possibility of deadlocks exist.

As various `Cairo` functions can acquire devices when called, these functions may also cause deadlocks when you call them with an acquired device. So you must not have a device acquired when calling them. These functions are marked in the documentation.

### 8.3.5.2 `cobj()` [1/2]

```
cobject * Cairo::Device::cobj () [inline]
```

### 8.3.5.3 `cobj()` [2/2]

```
const cobject * Cairo::Device::cobj() const [inline]
```

### 8.3.5.4 `finish()`

```
void Cairo::Device::finish ()
```

This function finishes the device and drops all references to external resources.

All surfaces, fonts and other objects created for this device will be finished, too. Further operations on the device will not affect the device but will instead trigger a `DEVICE_FINISHED` error.

When the last reference to the device is dropped, cairo will call `finish()` if it hasn't been called already, before freeing the resources associated with the device.

This function may acquire devices.

#### 8.3.5.5 flush()

```
void Cairo::Device::flush ()
```

Finish any pending operations for the device and also restore any temporary modifications cairo has made to the device's state.

This function must be called before switching from using the device with [Cairo](#) to operating on it directly with native APIs. If the device doesn't support direct access, then this function does nothing.

This function may acquire devices.

#### 8.3.5.6 get\_type()

```
DeviceType Cairo::Device::get_type () const
```

This function returns the type of the device.

#### 8.3.5.7 reference()

```
void Cairo::Device::reference () const
```

#### 8.3.5.8 release()

```
void Cairo::Device::release ()
```

Releases a device previously acquired using [acquire\(\)](#).

#### 8.3.5.9 unreference()

```
void Cairo::Device::unreference () const
```

### 8.3.6 Member Data Documentation

#### 8.3.6.1 m\_cobject

```
cobject* Cairo::Device::m_cobject [protected]
```

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

- caiomm/device.h

## 8.4 Cairo::Device::Lock Class Reference

A convenience class for acquiring a [Device](#) object in an exception-safe manner.

```
#include <caiomm/device.h>
```

## Public Member Functions

- [Lock](#) (const [RefPtr](#)< [Device](#) > &device)  
*Create a new [Device](#) lock for device.*
- [Lock](#) (const [Lock](#) &other)
- [~Lock](#) ()

### 8.4.1 Detailed Description

A convenience class for acquiring a [Device](#) object in an exception-safe manner.

The device is automatically acquired when a [Lock](#) object is created and released when the [Lock](#) object is destroyed. For example:

```
void
my_device_modifying_function (const RefPtr<Device>& device)
{
    // Ensure the device is properly reset
    device->flush();

    Device::Lock lock(device);
    // Do the custom operations on the device here.
    // But do not call any Cairo functions that might acquire devices.

} // device is automatically released at the end of the function scope
```

### 8.4.2 Constructor & Destructor Documentation

#### 8.4.2.1 Lock() [1/2]

```
Cairo::Device::Lock::Lock (
    const RefPtr< Device > & device)
```

Create a new [Device](#) lock for *device*.

#### 8.4.2.2 Lock() [2/2]

```
Cairo::Device::Lock::Lock (
    const Lock & other)
```

#### 8.4.2.3 ~Lock()

```
Cairo::Device::Lock::~~Lock ()
```

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

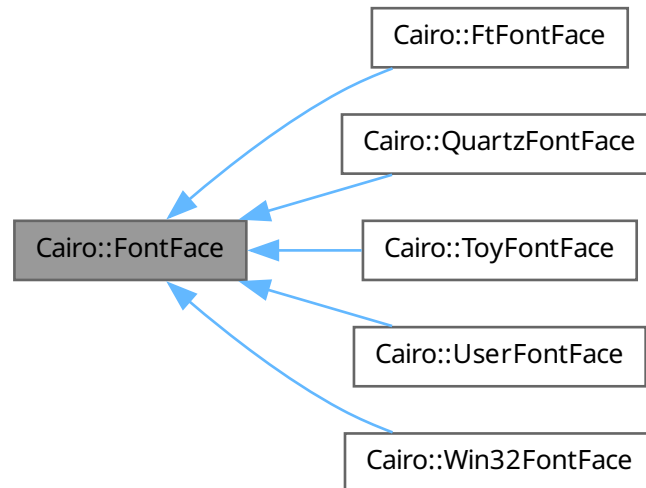
- cairomm/device.h

## 8.5 Cairo::FontFace Class Reference

A [FontFace](#) represents a particular font at a particular weight, slant, and other characteristic but no size, transformation, or size.

```
#include <cairomm/fontface.h>
```

Inheritance diagram for Cairo::FontFace:



### Public Types

- typedef cairo\_font\_face\_t [cobject](#)

### Public Member Functions

- [FontFace](#) (cairo\_font\_face\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [FontFace](#) (const [FontFace](#) &)=delete
- [FontFace](#) & operator= (const [FontFace](#) &)=delete
- virtual ~[FontFace](#) ()
- [FontType](#) get\_type () const  
*Returns the type of the backend used to create a font face.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

### Protected Attributes

- [cobject](#) \* [m\\_cobject](#)

### 8.5.1 Detailed Description

A [FontFace](#) represents a particular font at a particular weight, slant, and other characteristic but no size, transformation, or size.

Font faces are created using font-backend-specific constructors or implicitly using the toy text API by way of [Context::select\\_font\\_face\(\)](#). The resulting face can be accessed using [Context::get\\_font\\_face\(\)](#).

### 8.5.2 Member Typedef Documentation

#### 8.5.2.1 cobject

```
typedef cairo_font_face_t Cairo::FontFace::cobject
```

### 8.5.3 Constructor & Destructor Documentation

#### 8.5.3.1 FontFace() [1/2]

```
Cairo::FontFace::FontFace (
    cairo_font_face_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

##### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

#### 8.5.3.2 FontFace() [2/2]

```
Cairo::FontFace::FontFace (
    const FontFace & ) [delete]
```

#### 8.5.3.3 ~FontFace()

```
virtual Cairo::FontFace::~~FontFace () [virtual]
```

### 8.5.4 Member Function Documentation

#### 8.5.4.1 cobj() [1/2]

```
cobject * Cairo::FontFace::cobj () [inline]
```

#### 8.5.4.2 cobj() [2/2]

```
const cobject * Cairo::FontFace::cobj () const [inline]
```

#### 8.5.4.3 get\_type()

```
FontType Cairo::FontFace::get_type () const
```

Returns the type of the backend used to create a font face.

#### 8.5.4.4 operator=()

```
FontFace & Cairo::FontFace::operator= (  
    const FontFace & ) [delete]
```

#### 8.5.4.5 reference()

```
void Cairo::FontFace::reference () const
```

#### 8.5.4.6 unreference()

```
void Cairo::FontFace::unreference () const
```

### 8.5.5 Member Data Documentation

#### 8.5.5.1 m\_cobject

```
cobject* Cairo::FontFace::m_cobject [protected]
```

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

- cairomm/fontface.h

## 8.6 Cairo::FontOptions Class Reference

The font options specify how fonts should be rendered.

```
#include <cairomm/fontoptions.h>
```

## Public Types

- enum class [HintStyle](#) {  
[DEFAULT](#) = CAIRO\_HINT\_STYLE\_DEFAULT ,  
[NONE](#) = CAIRO\_HINT\_STYLE\_NONE ,  
[SLIGHT](#) = CAIRO\_HINT\_STYLE\_SLIGHT ,  
[MEDIUM](#) = CAIRO\_HINT\_STYLE\_MEDIUM ,  
[FULL](#) = CAIRO\_HINT\_STYLE\_FULL }  
*Specifies the type of hinting to do on font outlines.*
- enum class [HintMetrics](#) {  
[DEFAULT](#) = CAIRO\_HINT\_METRICS\_DEFAULT ,  
[OFF](#) = CAIRO\_HINT\_METRICS\_OFF ,  
[ON](#) = CAIRO\_HINT\_METRICS\_ON }  
*Specifies whether to hint font metrics; hinting font metrics means quantizing them so that they are integer values in device space.*
- typedef cairo\_font\_options\_t [cobject](#)

## Public Member Functions

- [FontOptions](#) ()
- [FontOptions](#) (cairo\_font\_options\_t \*[cobject](#), bool take\_ownership=false)
- [FontOptions](#) (const [FontOptions](#) &src)
- virtual ~[FontOptions](#) ()
- [FontOptions](#) & [operator=](#) (const [FontOptions](#) &src)
- bool [operator==](#) (const [FontOptions](#) &src) const
- void [merge](#) (const [FontOptions](#) &other)  
*Merges non-default options from other into this, replacing existing values.*
- unsigned long [hash](#) () const  
*Compute a hash for the font options object; this value will be useful when storing an object containing a [FontOptions](#) in a hash table.*
- void [set\\_antialias](#) ([Antialias](#) antialias)  
*Sets the antialiasing mode for the font options object.*
- [Antialias](#) [get\\_antialias](#) () const  
*Gets the antialiasing mode for the font options object.*
- void [set\\_subpixel\\_order](#) ([SubpixelOrder](#) subpixel\_order)  
*Sets the subpixel order for the font options object.*
- [SubpixelOrder](#) [get\\_subpixel\\_order](#) () const  
*Gets the subpixel order for the font options object.*
- void [set\\_hint\\_style](#) ([HintStyle](#) hint\_style)  
*Sets the hint style for font outlines for the font options object.*
- [HintStyle](#) [get\\_hint\\_style](#) () const  
*Gets the hint style for font outlines for the font options object.*
- void [set\\_hint\\_metrics](#) ([HintMetrics](#) hint\_metrics)  
*Sets the metrics hinting mode for the font options object.*
- [HintMetrics](#) [get\\_hint\\_metrics](#) () const  
*Gets the metrics hinting mode for the font options object.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const

## Protected Attributes

- [cobject](#) \* [m\\_cobject](#)



### 8.6.1 Detailed Description

The font options specify how fonts should be rendered.

Most of the time the font options implied by a surface are just right and do not need any changes, but for pixel-based targets tweaking font options may result in superior output on a particular display.

### 8.6.2 Member Typedef Documentation

#### 8.6.2.1 cobject

```
typedef cairo_font_options_t Cairo::FontOptions::cobject
```

### 8.6.3 Member Enumeration Documentation

#### 8.6.3.1 HintMetrics

```
enum class Cairo::FontOptions::HintMetrics [strong]
```

Specifies whether to hint font metrics; hinting font metrics means quantizing them so that they are integer values in device space.

Doing this improves the consistency of letter and line spacing, however it also means that text will be laid out differently at different zoom factors.

Enumerator

DEFAULT	Hint metrics in the default manner for the font backend and target device.
OFF	Do not hint font metrics.
ON	Hint font metrics.

#### 8.6.3.2 HintStyle

```
enum class Cairo::FontOptions::HintStyle [strong]
```

Specifies the type of hinting to do on font outlines.

Hinting is the process of fitting outlines to the pixel grid in order to improve the appearance of the result. Since hinting outlines involves distorting them, it also reduces the faithfulness to the original outline shapes. Not all of the outline hinting styles are supported by all font backends.

New entries may be added in future versions.

Enumerator

DEFAULT	Use the default hint style for font backend and target device.
NONE	Do not hint outlines.
SLIGHT	Hint outlines slightly to improve contrast while retaining food fidelity to the original shapes.
MEDIUM	Hint outlines with medium strength giving a compromise between fidelity to the original shapes and contrast.
FULL	Hint outlines to maximize contrast.

## 8.6.4 Constructor & Destructor Documentation

### 8.6.4.1 FontOptions() [1/3]

```
Cairo::FontOptions::FontOptions ()
```

### 8.6.4.2 FontOptions() [2/3]

```
Cairo::FontOptions::FontOptions (
    cairo_font_options_t * cobject,
    bool take_ownership = false) [explicit]
```

### 8.6.4.3 FontOptions() [3/3]

```
Cairo::FontOptions::FontOptions (
    const FontOptions & src)
```

### 8.6.4.4 ~FontOptions()

```
virtual Cairo::FontOptions::~FontOptions () [virtual]
```

## 8.6.5 Member Function Documentation

### 8.6.5.1 cobj() [1/2]

```
cobject * Cairo::FontOptions::cobj () [inline]
```

### 8.6.5.2 cobj() [2/2]

```
const cobject * Cairo::FontOptions::cobj () const [inline]
```

### 8.6.5.3 get\_antialias()

```
Antialias Cairo::FontOptions::get_antialias () const
```

Gets the antialiasing mode for the font options object.

#### Returns

the antialiasing mode

#### 8.6.5.4 get\_hint\_metrics()

```
HintMetrics Cairo::FontOptions::get_hint_metrics () const
```

Gets the metrics hinting mode for the font options object.

See the documentation for HintMetrics for full details.

Return value: the metrics hinting mode for the font options object.

#### 8.6.5.5 get\_hint\_style()

```
HintStyle Cairo::FontOptions::get_hint_style () const
```

Gets the hint style for font outlines for the font options object.

See the documentation for HintStyle for full details.

##### Returns

the hint style for the font options object.

#### 8.6.5.6 get\_subpixel\_order()

```
SubpixelOrder Cairo::FontOptions::get_subpixel_order () const
```

Gets the subpixel order for the font options object.

See the documentation for SubpixelOrder for full details.

##### Returns

the subpixel order for the font options object.

#### 8.6.5.7 hash()

```
unsigned long Cairo::FontOptions::hash () const
```

Compute a hash for the font options object; this value will be useful when storing an object containing a [FontOptions](#) in a hash table.

##### Returns

the hash value for the font options object. The return value can be cast to a 32-bit type if a 32-bit hash value is needed.

#### 8.6.5.8 merge()

```
void Cairo::FontOptions::merge (  
    const FontOptions & other)
```

Merges non-default options from *other* into this, replacing existing values.

This operation can be thought of as somewhat similar to compositing *other* onto this with the operation of OPERATION\_OVER.

## Parameters

<i>other</i>	another <a href="#">FontOptions</a>
--------------	-------------------------------------

**8.6.5.9 operator=()**

```
FontOptions & Cairo::FontOptions::operator= (
    const FontOptions & src)
```

**8.6.5.10 operator==()**

```
bool Cairo::FontOptions::operator== (
    const FontOptions & src) const
```

**8.6.5.11 set\_antialias()**

```
void Cairo::FontOptions::set_antialias (
    Antialias antialias)
```

Sets the antialiasing mode for the font options object.

This specifies the type of antialiasing to do when rendering text.

## Parameters

<i>antialias</i>	the new antialiasing mode.
------------------	----------------------------

**8.6.5.12 set\_hint\_metrics()**

```
void Cairo::FontOptions::set_hint_metrics (
    HintMetrics hint_metrics)
```

Sets the metrics hinting mode for the font options object.

This controls whether metrics are quantized to integer values in device units. See the documentation for [HintMetrics](#) for full details.

## Parameters

<i>hint_metrics</i>	the new metrics hinting mode.
---------------------	-------------------------------

**8.6.5.13 set\_hint\_style()**

```
void Cairo::FontOptions::set_hint_style (
    HintStyle hint_style)
```

Sets the hint style for font outlines for the font options object.

This controls whether to fit font outlines to the pixel grid, and if so, whether to optimize for fidelity or contrast. See the documentation for [HintStyle](#) for full details.

## Parameters

<i>hint_style</i>	the new hint style.
-------------------	---------------------

**8.6.5.14 set\_subpixel\_order()**

```
void Cairo::FontOptions::set_subpixel_order (
    SubpixelOrder subpixel_order)
```

Sets the subpixel order for the font options object.

The subpixel order specifies the order of color elements within each pixel on the display device when rendering with an antialiasing mode of [Cairo::ANTIALIAS\\_SUBPIXEL](#). See the documentation for SubpixelOrder for full details.

## Parameters

<i>subpixel_order</i>	the new subpixel order.
-----------------------	-------------------------

**8.6.6 Member Data Documentation****8.6.6.1 m\_cobject**

```
cobject* Cairo::FontOptions::m_cobject [protected]
```

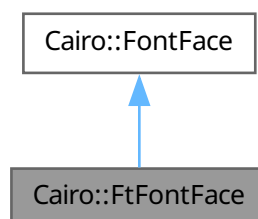
The documentation for this class was generated from the following file:

- cairomm/fontoptions.h

**8.7 Cairo::FtFontFace Class Reference**

```
#include <cairomm/fontface.h>
```

Inheritance diagram for Cairo::FtFontFace:



## Public Member Functions

- void [set\\_synthesize](#) ([FtSynthesize](#) synth\_flags)  
*Sets synthesis options to control how FreeType renders the glyphs for a particular font face.*
- void [unset\\_synthesize](#) ([FtSynthesize](#) synth\_flags)  
*Unsets the specified FreeType glyph synthesis options.*
- [FtSynthesize get\\_synthesize](#) () const  
*Returns currently active FreeType glyph synthesis options.*

## Public Member Functions inherited from [Cairo::FontFace](#)

- [FontFace](#) (cairo\_font\_face\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [FontFace](#) (const [FontFace](#) &)=delete
- [FontFace](#) & [operator=](#) (const [FontFace](#) &)=delete
- virtual [~FontFace](#) ()
- [FontType get\\_type](#) () const  
*Returns the type of the backend used to create a font face.*
- [cobject \\* cobj](#) ()
- const [cobject \\* cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Static Public Member Functions

- static [RefPtr< FtFontFace > create](#) (FT\_Face face, int load\_flags)  
*Creates a new font face for the FreeType font backend from a pre-opened FreeType face.*

## Protected Member Functions

- [FtFontFace](#) (FT\_Face face, int load\_flags)

## Additional Inherited Members

## Public Types inherited from [Cairo::FontFace](#)

- typedef cairo\_font\_face\_t [cobject](#)

## Protected Attributes inherited from [Cairo::FontFace](#)

- [cobject \\* m\\_cobject](#)

## 8.7.1 Constructor & Destructor Documentation

### 8.7.1.1 FtFontFace()

```
Cairo::FtFontFace::FtFontFace (
    FT_Face face,
    int load_flags) [protected]
```

## 8.7.2 Member Function Documentation

### 8.7.2.1 create()

```
static RefPtr< FtFontFace > Cairo::FtFontFace::create (
    FT_Face face,
    int load_flags) [static]
```

Creates a new font face for the FreeType font backend from a pre-opened FreeType face.

This font can then be used with [Context::set\\_font\\_face\(\)](#) or [FtScaledFont::create\(\)](#).

As an example, here is how one might correctly couple the lifetime of the FreeType face object to the [FtFontFace](#):

```
static const cairo_user_data_key_t key;

font_face = cairo_ft_font_face_create_for_ft_face (ft_face, 0);
status = cairo_font_face_set_user_data (font_face, &key,
                                         ft_face, (cairo_destroy_func_t) FT_Done_Face);
if (status) {
    cairo_font_face_destroy (font_face);
    FT_Done_Face (ft_face);
    return ERROR;
}
```

#### Parameters

<i>face</i>	A FreeType face object, already opened. This must be kept around until the face's ref_count drops to zero and it is freed. Since the face may be referenced internally to <a href="#">Cairo</a> , the best way to determine when it is safe to free the face is to pass a <code>cairo_destroy_func_t</code> to <code>cairo_font_face_set_user_data()</code> .
<i>load_flags</i>	flags to pass to <code>FT_Load_Glyph</code> when loading glyphs from the font. These flags are OR'ed together with the flags derived from the <code>cairo_font_options_t</code> passed to <code>cairo_scaled_font_create()</code> , so only a few values such as <code>FT_LOAD_VERTICAL_LAYOUT</code> , and <code>FT_LOAD_FORCE_AUTOHINT</code> are useful. You should not pass any of the flags affecting the load target, such as <code>FT_LOAD_TARGET_LIGHT</code> .

Since

1.8

### 8.7.2.2 get\_synthesize()

```
FtSynthesize Cairo::FtFontFace::get_synthesize () const
```

Returns currently active FreeType glyph synthesis options.

Since

1.12

### 8.7.2.3 set\_synthesize()

```
void Cairo::FtFontFace::set_synthesize (
    FtSynthesize synth_flags)
```

Sets synthesis options to control how FreeType renders the glyphs for a particular font face.

The given options are ORed with the currently active options.

## Parameters

<i>synth_flags</i>	A set of synthesis options to enable
--------------------	--------------------------------------

## Since

1.12

**8.7.2.4 unset\_synthesize()**

```
void Cairo::FtFontFace::unset_synthesize (
    FtSynthesize synth_flags)
```

Unsets the specified FreeType glyph synthesis options.

## Parameters

<i>synth_flags</i>	A set of synthesis options to disable
--------------------	---------------------------------------

## Since

1.12

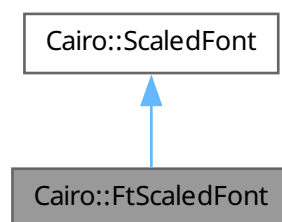
The documentation for this class was generated from the following file:

- cairomm/fontface.h

**8.8 Cairo::FtScaledFont Class Reference**

```
#include <cairomm/scaledfont.h>
```

Inheritance diagram for Cairo::FtScaledFont:





## Public Member Functions

- FT\_Face [lock\\_face](#) ()  
*Gets the FT\_Face object from a FreeType backend font and scales it appropriately for the font.*
- void [unlock\\_face](#) ()  
*Releases a face obtained with [lock\\_face\(\)](#).*

## Public Member Functions inherited from Cairo::ScaledFont

- [cobject](#) \* [cobj](#) ()  
*Provides acces to the underlying C cairo object.*
- const [cobject](#) \* [cobj](#) () const  
*Provides acces to the underlying C cairo object.*
- [ScaledFont](#) ([cobject](#) \*[cobj](#), bool has\_reference=false)  
*Create a C++ wrapper object from the C instance.*
- [ScaledFont](#) (const [ScaledFont](#) &)=delete
- [ScaledFont](#) & [operator=](#) (const [ScaledFont](#) &)=delete
- virtual ~[ScaledFont](#) ()
- void [get\\_extents](#) ([FontExtents](#) &extents) const  
*Gets the metrics for a [ScaledFont](#).*
- void [get\\_text\\_extents](#) (const **std::string** &utf8, [TextExtents](#) &extents) const  
*Gets the extents for a string of text.*
- void [get\\_glyph\\_extents](#) (const **std::vector**< [Glyph](#) > &glyphs, [TextExtents](#) &extents) const  
*Gets the extents for an array of glyphs.*
- [RefPtr](#)< [FontFace](#) > [get\\_font\\_face](#) () const  
*The [FontFace](#) with which this [ScaledFont](#) was created.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Gets the [FontOptions](#) with which the [ScaledFont](#) was created.*
- void [get\\_font\\_matrix](#) ([Matrix](#) &font\_matrix) const  
*Gets the font matrix with which the [ScaledFont](#) was created.*
- void [get\\_ctm](#) ([Matrix](#) &ctm) const  
*Gets the CTM with which the [ScaledFont](#) was created.*
- [FontType](#) [get\\_type](#) () const  
*Gets the type of scaled Font.*
- void [text\\_to\\_glyphs](#) (double x, double y, const **std::string** &utf8, **std::vector**< [Glyph](#) > &glyphs, **std::vector**< [TextCluster](#) > &clusters, [TextClusterFlags](#) &cluster\_flags)  
*Converts a string of text to glyphs and clusters.*
- void [get\\_scale\\_matrix](#) ([Matrix](#) &scale\_matrix) const  
*Stores the scale matrix of this scaled font into matrix.*

## Static Public Member Functions

- static [RefPtr](#)< [FtScaledFont](#) > [create](#) (const [RefPtr](#)< [FtFontFace](#) > &font\_face, const [Matrix](#) &font\_matrix, const [Matrix](#) &ctm, const [FontOptions](#) &options=[FontOptions](#)())  
*Creates a [ScaledFont](#) From a [FtFontFace](#).*

## Static Public Member Functions inherited from Cairo::ScaledFont

- static [RefPtr](#)< [ScaledFont](#) > [create](#) (const [RefPtr](#)< [FontFace](#) > &font\_face, const [Matrix](#) &font\_matrix, const [Matrix](#) &ctm, const [FontOptions](#) &options=[FontOptions](#)())  
*Creates a [ScaledFont](#) object from a font face and matrices that describe the size of the font and the environment in which it will be used.*

### Protected Member Functions

- `FtScaledFont` (const `RefPtr< FtFontFace >` &font\_face, const `Matrix` &font\_matrix, const `Matrix` &ctm, const `FontOptions` &options=`FontOptions`())

### Protected Member Functions inherited from `Cairo::ScaledFont`

- `ScaledFont` (const `RefPtr< FontFace >` &font\_face, const `Matrix` &font\_matrix, const `Matrix` &ctm, const `FontOptions` &options=`FontOptions`())

### Additional Inherited Members

### Public Types inherited from `Cairo::ScaledFont`

- typedef `cairo_scaled_font_t` `cobject`  
*The underlying C cairo object type.*

### Protected Attributes inherited from `Cairo::ScaledFont`

- `cobject * m_cobject`  
*The underlying C cairo object that is wrapped by this `ScaledFont`.*

## 8.8.1 Detailed Description

Since

1.8

## 8.8.2 Constructor & Destructor Documentation

### 8.8.2.1 `FtScaledFont()`

```
Cairo::FtScaledFont::FtScaledFont (
    const RefPtr< FtFontFace > & font_face,
    const Matrix & font_matrix,
    const Matrix & ctm,
    const FontOptions & options = FontOptions()) [protected]
```

## 8.8.3 Member Function Documentation

### 8.8.3.1 `create()`

```
static RefPtr< FtScaledFont > Cairo::FtScaledFont::create (
    const RefPtr< FtFontFace > & font_face,
    const Matrix & font_matrix,
    const Matrix & ctm,
    const FontOptions & options = FontOptions()) [static]
```

Creates a `ScaledFont` From a `FtFontFace`.

Since

1.8

### 8.8.3.2 lock\_face()

```
FT_Face Cairo::FtScaledFont::lock_face ()
```

Gets the FT\_Face object from a FreeType backend font and scales it appropriately for the font.

You must release the face with [unlock\\_face\(\)](#) when you are done using it. Since the FT\_Face object can be shared between multiple [ScaledFont](#) objects, you must not lock any other font objects until you unlock this one. A count is kept of the number of times [lock\\_face\(\)](#) is called. [unlock\\_face\(\)](#) must be called the same number of times.

You must be careful when using this function in a library or in a threaded application, because freetype's design makes it unsafe to call freetype functions simultaneously from multiple threads, (even if using distinct FT\_Face objects). Because of this, application code that acquires an FT\_Face object with this call must add it's own locking to protect any use of that object, (and which also must protect any other calls into cairo as almost any cairo function might result in a call into the freetype library).

#### Returns

The FT\_Face object for font, scaled appropriately, or NULL if scaled\_font is in an error state or there is insufficient memory.

#### Since

1.8

### 8.8.3.3 unlock\_face()

```
void Cairo::FtScaledFont::unlock_face ()
```

Releases a face obtained with [lock\\_face\(\)](#).

#### Since

1.8

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

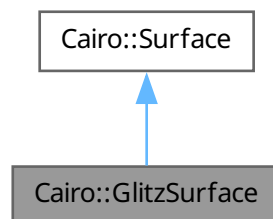
- cairomm/scaledfont.h

## 8.9 Cairo::GlitzSurface Class Reference

A GlitzSurface provides a way to render to the X Window System using Glitz.

```
#include <cairomm/surface.h>
```

Inheritance diagram for Cairo::GlitzSurface:



## Public Member Functions

- [GlitzSurface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~GlitzSurface](#) () override

## Public Member Functions inherited from [Cairo::Surface](#)

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & operator= (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* data, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type get\\_type](#) () const
- [Content get\\_content](#) () const

*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*

- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const  
*Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)  
*Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)  
*Writes the [Surface](#) to the write function.*
- [RefPtr< Device >](#) [get\\_device](#) ()  
*This function returns the device for a surface.*
- [cobject \\* cobj](#) ()  
*Provides acces to the underlying C cairo surface.*
- const [cobject \\* cobj](#) () const  
*Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr< GlitzSurface >](#) [create](#) (glitz\_surface\_t \*surface)  
*Creates a new [GlitzSurface](#).*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) other, [Content](#) content, int width, int height)  
*Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)  
*Create a new surface that is a rectangle within the target surface.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {  
[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,  
[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,  
[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,  
[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,  
[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,  
[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,  
[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,  
[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,

```

WIN32_PRINTING = CAIRO_SURFACE_TYPE_WIN32_PRINTING ,
QUARTZ_IMAGE = CAIRO_SURFACE_TYPE_QUARTZ_IMAGE ,
SCRIPT = CAIRO_SURFACE_TYPE_SCRIPT ,
QT = CAIRO_SURFACE_TYPE_QT ,
RECORDING = CAIRO_SURFACE_TYPE_RECORDING ,
VG = CAIRO_SURFACE_TYPE_VG ,
GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)

*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*

- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)

*This is the type of function which is called when a backend needs to read data from an input stream.*

- typedef sigc::slot< void()> [SlotDestroy](#)

*For instance, void on\_destroy();.*

- typedef cairo\_surface\_t [cobject](#)

*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)

*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.9.1 Detailed Description

A GlitzSurface provides a way to render to the X Window System using Glitz.

This provides a way to use OpenGL-accelerated graphics from cairo. If you want to use hardware-accelerated graphics within the X Window system, you should use this [Surface](#) type.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with Glitz support

#### Warning

This is an experimental surface. It is not yet marked as a fully supported surface by the cairo library

## 8.9.2 Constructor & Destructor Documentation

### 8.9.2.1 GlitzSurface()

```
Cairo::GlitzSurface::GlitzSurface (  
    cairo_surface_t * cobject,  
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.9.2.2 ~GlitzSurface()

```
Cairo::GlitzSurface::~~GlitzSurface () [override]
```

## 8.9.3 Member Function Documentation

### 8.9.3.1 create()

```
static RefPtr< GlitzSurface > Cairo::GlitzSurface::create (  
    glitz_surface_t * surface) [static]
```

Creates a new [GlitzSurface](#).

#### Parameters

<i>surface</i>	a glitz surface type
----------------	----------------------

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

- cairomm/surface.h

## 8.10 Cairo::Gradient Class Reference

```
#include <cairomm/pattern.h>
```

Inheritance diagram for Cairo::Gradient:



### Public Member Functions

- [Gradient](#) (cairo\_pattern\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~Gradient](#) () override
- void [add\\_color\\_stop\\_rgb](#) (double offset, double red, double green, double blue)  
*Adds an opaque color stop to a gradient pattern.*
- void [add\\_color\\_stop\\_rgba](#) (double offset, double red, double green, double blue, double alpha)  
*Adds a translucent color stop to a gradient pattern.*
- **std::vector**< [ColorStop](#) > [get\\_color\\_stops](#) () const  
*Gets the color stops and offsets for this [Gradient](#).*

### Public Member Functions inherited from [Cairo::Pattern](#)

- [Pattern](#) (cairo\_pattern\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Pattern](#) (const [Pattern](#) &)=delete
- [Pattern](#) & [operator=](#) (const [Pattern](#) &)=delete
- virtual [~Pattern](#) ()
- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the pattern's transformation matrix to @matrix.*
- void [get\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the pattern's transformation matrix.*
- [Matrix](#) [get\\_matrix](#) () const  
*Returns the pattern's transformation matrix.*
- [Type](#) [get\\_type](#) () const



- Returns the type of the pattern.*
- void [set\\_extend](#) ([Extend](#) extend)
- Sets the mode to be used for drawing outside the area of a pattern.*
- [Extend](#) [get\\_extend](#) () const
- Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

### Protected Member Functions

- [Gradient](#) ()

### Protected Member Functions inherited from [Cairo::Pattern](#)

- [Pattern](#) ()

### Additional Inherited Members

### Public Types inherited from [Cairo::Pattern](#)

- enum class [Type](#) {  
[SOLID](#) = CAIRO\_PATTERN\_TYPE\_SOLID ,  
[SURFACE](#) = CAIRO\_PATTERN\_TYPE\_SURFACE ,  
[LINEAR](#) = CAIRO\_PATTERN\_TYPE\_LINEAR ,  
[RADIAL](#) = CAIRO\_PATTERN\_TYPE\_RADIAL }  
*Type is used to describe the type of a given pattern.*
- enum class [Extend](#) {  
[NONE](#) = CAIRO\_EXTEND\_NONE ,  
[REPEAT](#) = CAIRO\_EXTEND\_REPEAT ,  
[REFLECT](#) = CAIRO\_EXTEND\_REFLECT ,  
[PAD](#) = CAIRO\_EXTEND\_PAD }  
*Cairo::Extend is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).*
- typedef cairo\_pattern\_t [cobject](#)

### Protected Attributes inherited from [Cairo::Pattern](#)

- [cobject](#) \* [m\\_cobject](#)

## 8.10.1 Constructor & Destructor Documentation

### 8.10.1.1 Gradient() [1/2]

```
Cairo::Gradient::Gradient (
    cairo_pattern_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

**8.10.1.2 ~Gradient()**

```
Cairo::Gradient::~~Gradient () [override]
```

**8.10.1.3 Gradient() [2/2]**

```
Cairo::Gradient::Gradient () [protected]
```

**8.10.2 Member Function Documentation****8.10.2.1 add\_color\_stop\_rgb()**

```
void Cairo::Gradient::add_color_stop_rgb (
    double offset,
    double red,
    double green,
    double blue)
```

Adds an opaque color stop to a gradient pattern.

The offset specifies the location along the gradient's control vector. For example, a linear gradient's control vector is from (x0,y0) to (x1,y1) while a radial gradient's control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in [Context::set\\_source\\_rgb\(\)](#).

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added, (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

## Parameters

<i>offset</i>	an offset in the range [0.0 .. 1.0]
<i>red</i>	red component of color
<i>green</i>	green component of color
<i>blue</i>	blue component of color

### 8.10.2.2 add\_color\_stop\_rgba()

```
void Cairo::Gradient::add_color_stop_rgba (
    double offset,
    double red,
    double green,
    double blue,
    double alpha)
```

Adds a translucent color stop to a gradient pattern.

The offset specifies the location along the gradient's control vector. For example, a linear gradient's control vector is from (x0,y0) to (x1,y1) while a radial gradient's control vector is from any point on the start circle to the corresponding point on the end circle.

The color is specified in the same way as in [Context::set\\_source\\_rgba\(\)](#).

If two (or more) stops are specified with identical offset values, they will be sorted according to the order in which the stops are added, (stops added earlier will compare less than stops added later). This can be useful for reliably making sharp color transitions instead of the typical blend.

#### Parameters

<i>offset</i>	an offset in the range [0.0 .. 1.0]
<i>red</i>	red component of color
<i>green</i>	green component of color
<i>blue</i>	blue component of color
<i>alpha</i>	alpha component of color

### 8.10.2.3 get\_color\_stops()

```
std::vector< ColorStop > Cairo::Gradient::get_color_stops () const
```

Gets the color stops and offsets for this [Gradient](#).

Since

1.4

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

- caiomm/pattern.h

## 8.11 Cairo::ImageSurface Class Reference

Image surfaces provide the ability to render to memory buffers either allocated by cairo or by the calling code.

```
#include <caiomm/surface.h>
```

Inheritance diagram for Cairo::ImageSurface:



### Public Member Functions

- [ImageSurface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~ImageSurface](#) () override
- int [get\\_width](#) () const  
*Gets the width of the [ImageSurface](#) in pixels.*
- int [get\\_height](#) () const  
*Gets the height of the [ImageSurface](#) in pixels.*
- [Format get\\_format](#) () const  
*Gets the format of the surface.*
- int [get\\_stride](#) () const  
*Returns the stride of the image surface in bytes (or 0 if surface is not an image surface).*
  
- unsigned char \* [get\\_data](#) ()  
*Get a pointer to the data of the image surface, for direct inspection or modification.*
- const unsigned char \* [get\\_data](#) () const

## Public Member Functions inherited from Cairo::Surface

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & [operator=](#) (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* **data**, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const

- Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.
- void [write\\_to\\_png](#) (const **std::string** &filename)

Writes the contents of surface to a new file filename as a PNG image.
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)

Writes the [Surface](#) to the write function.
- [RefPtr](#)< [Device](#) > [get\\_device](#) ()

This function returns the device for a surface.
- [cobject](#) \* [cobj](#) ()

Provides acces to the underlying C cairo surface.
- const [cobject](#) \* [cobj](#) () const

Provides acces to the underlying C cairo surface.

### Static Public Member Functions

- static int [format\\_stride\\_for\\_width](#) ([Format](#) format, int width)

This function provides a stride value that will respect all alignment requirements of the accelerated image-rendering code within cairo.
- static [RefPtr](#)< [ImageSurface](#) > [create](#) ([Format](#) format, int width, int height)

Creates an image surface of the specified format and dimensions.
- static [RefPtr](#)< [ImageSurface](#) > [create](#) (unsigned char \* **data**, [Format](#) format, int width, int height, int **stride**)

Creates an image surface for the provided pixel data.
- static [RefPtr](#)< [ImageSurface](#) > [create\\_from\\_png](#) ( **std::string** filename)

Creates a new image surface and initializes the contents to the given PNG file.
- static [RefPtr](#)< [ImageSurface](#) > [create\\_from\\_png\\_stream](#) (const [SlotReadFunc](#) &read\_func)

Creates a new image surface from PNG data read incrementally via the read\_func function.

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > other, [Content](#) content, int width, int height)

Create a new surface that is as compatible as possible with an existing surface.
- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > &target, double x, double y, double width, double height)

Create a new surface that is a rectangle within the target surface.

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {

[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,

[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,

[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,

[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,

[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,

[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,

[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,

[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,

[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,

[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,

[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,

```

SVG = CAIRO_SURFACE_TYPE_SVG ,
OS2 = CAIRO_SURFACE_TYPE_OS2 ,
WIN32_PRINTING = CAIRO_SURFACE_TYPE_WIN32_PRINTING ,
QUARTZ_IMAGE = CAIRO_SURFACE_TYPE_QUARTZ_IMAGE ,
SCRIPT = CAIRO_SURFACE_TYPE_SCRIPT ,
QT = CAIRO_SURFACE_TYPE_QT ,
RECORDING = CAIRO_SURFACE_TYPE_RECORDING ,
VG = CAIRO_SURFACE_TYPE_VG ,
GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)  
*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*
- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)  
*This is the type of function which is called when a backend needs to read data from an input stream.*
- typedef sigc::slot< void()> [SlotDestroy](#)  
*For instance, `void on_destroy();`.*
- typedef cairo\_surface\_t [cobject](#)  
*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)  
*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.11.1 Detailed Description

Image surfaces provide the ability to render to memory buffers either allocated by cairo or by the calling code.

The supported image formats are those defined in [Format](#)

An [ImageSurface](#) is the most generic type of [Surface](#) and the only one that is available by default. You can either create an [ImageSurface](#) whose data is managed by [Cairo](#), or you can create an [ImageSurface](#) with a data buffer that you allocated yourself so that you can have full access to the data.

When you create an [ImageSurface](#) with your own data buffer, you are free to examine the results at any point and do whatever you want with it. Note that if you modify anything and later want to continue to draw to the surface with cairo, you must let cairo know via [Cairo::Surface::mark\\_dirty\(\)](#)

Note that like all surfaces, an [ImageSurface](#) is a reference-counted object that should be used via [Cairo::RefPtr](#).

## 8.11.2 Constructor & Destructor Documentation

### 8.11.2.1 ImageSurface()

```
Cairo::ImageSurface::ImageSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.11.2.2 ~ImageSurface()

```
Cairo::ImageSurface::~ImageSurface () [override]
```

## 8.11.3 Member Function Documentation

### 8.11.3.1 create() [1/2]

```
static RefPtr< ImageSurface > Cairo::ImageSurface::create (
    Format format,
    int width,
    int height) [static]
```

Creates an image surface of the specified format and dimensions.

Initially the surface contents are all 0. (Specifically, within each pixel, each color or alpha channel belonging to format will be 0. The contents of bits within a pixel, but not belonging to the given format are undefined).

#### Parameters

<i>format</i>	format of pixels in the surface to create
<i>width</i>	width of the surface, in pixels
<i>height</i>	height of the surface, in pixels

#### Returns

a RefPtr to the newly created surface.

#### Examples

[image-surface.cc](#), [toy-text.cc](#), and [user-font.cc](#).



## 8.11.3.2 create() [2/2]

```
static RefPtr< ImageSurface > Cairo::ImageSurface::create (
    unsigned char * data,
    Format format,
    int width,
    int height,
    int stride) [static]
```

Creates an image surface for the provided pixel data.

The output buffer must be kept around until the surface is destroyed or [finish\(\)](#) is called on the surface. The initial contents of buffer will be used as the initial image contents; you must explicitly clear the buffer, using, for example, [Context::rectangle\(\)](#) and [Context::fill\(\)](#) if you want it cleared.

Note that the stride may be larger than  $width * bytes\_per\_pixel$  to provide proper alignment for each pixel and row. This alignment is required to allow high-performance rendering within cairo. The correct way to obtain a legal stride value is to call [format\\_stride\\_for\\_width\(\)](#) with the desired format and maximum image width value, and the use the resulting stride value to allocate the data and to create the image surface. See [format\\_stride\\_for\\_width\(\)](#) for example code.

## Parameters

<i>data</i>	a pointer to a buffer supplied by the application in which to write contents. This pointer must be suitably aligned for any kind of variable, (for example, a pointer returned by malloc).
<i>format</i>	the format of pixels in the buffer
<i>width</i>	the width of the image to be stored in the buffer
<i>height</i>	the height of the image to be stored in the buffer
<i>stride</i>	the number of bytes between the start of rows in the buffer as allocated. This value should always be computed by <a href="#">format_stride_for_width()</a> before allocating the data buffer.

## Returns

a RefPtr to the newly created surface.

## 8.11.3.3 create\_from\_png()

```
static RefPtr< ImageSurface > Cairo::ImageSurface::create_from_png (
    std::string filename) [static]
```

Creates a new image surface and initializes the contents to the given PNG file.

## Note

For this function to be available, cairo must have been compiled with PNG support.

## Parameters

<i>filename</i>	name of PNG file to load
-----------------	--------------------------

## Returns

a RefPtr to the new `cairo_surface_t` initialized with the contents of the PNG image file.

#### 8.11.3.4 create\_from\_png\_stream()

```
static RefPtr< ImageSurface > Cairo::ImageSurface::create_from_png_stream (
    const SlotReadFunc & read_func) [static]
```

Creates a new image surface from PNG data read incrementally via the `read_func` function.

##### Note

For this function to be available, cairo must have been compiled with PNG support.

##### Parameters

<i>read_func</i>	function called to read the data of the file
------------------	----------------------------------------------

##### Returns

a `RefPtr` to the new `cairo_surface_t` initialized with the contents of the PNG image file.

#### 8.11.3.5 format\_stride\_for\_width()

```
static int Cairo::ImageSurface::format_stride_for_width (
    Format format,
    int width) [static]
```

This function provides a stride value that will respect all alignment requirements of the accelerated image-rendering code within cairo.

Typical usage will be of the form:

```
int stride;
unsigned char *data;
Cairo::RefPtr<Cairo::ImageSurface> surface;

stride = Cairo::ImageSurface::format_stride_for_width (format, width);
data = malloc (stride * height);
surface = Cairo::ImageSurface::create (data, format, width, height);
```

##### Parameters

<i>format</i>	A Format value
<i>width</i>	The desired width of an image surface to be created.

##### Returns

the appropriate stride to use given the desired format and width, or -1 if either the format is invalid or the width too large.

##### Since

1.6

### 8.11.3.6 `get_data()` [1/2]

```
unsigned char * Cairo::ImageSurface::get_data ()
```

Get a pointer to the data of the image surface, for direct inspection or modification.

Return value: a pointer to the image data of this surface or NULL if @surface is not an image surface.

Since

1.2

### 8.11.3.7 `get_data()` [2/2]

```
const unsigned char * Cairo::ImageSurface::get_data () const
```

### 8.11.3.8 `get_format()`

```
Format Cairo::ImageSurface::get_format () const
```

Gets the format of the surface.

Since

1.2

### 8.11.3.9 `get_height()`

```
int Cairo::ImageSurface::get_height () const
```

Gets the height of the [ImageSurface](#) in pixels.

### 8.11.3.10 `get_stride()`

```
int Cairo::ImageSurface::get_stride () const
```

Returns the stride of the image surface in bytes (or 0 if surface is not an image surface).

The stride is the distance in bytes from the beginning of one row of the image data to the beginning of the next row.

Since

1.2

### 8.11.3.11 get\_width()

```
int Cairo::ImageSurface::get_width () const
```

Gets the width of the [ImageSurface](#) in pixels.

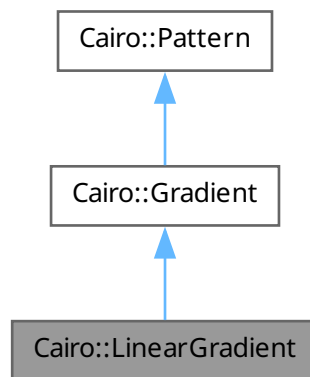
The documentation for this class was generated from the following file:

- cairomm/surface.h

## 8.12 Cairo::LinearGradient Class Reference

```
#include <cairomm/pattern.h>
```

Inheritance diagram for Cairo::LinearGradient:



### Public Member Functions

- [LinearGradient](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- void [get\\_linear\\_points](#) (double &x0, double &y0, double &x1, double &y1) const
- [~LinearGradient](#) () override

### Public Member Functions inherited from Cairo::Gradient

- [Gradient](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~Gradient](#) () override
- void [add\\_color\\_stop\\_rgb](#) (double offset, double red, double green, double blue)  
*Adds an opaque color stop to a gradient pattern.*
- void [add\\_color\\_stop\\_rgba](#) (double offset, double red, double green, double blue, double alpha)  
*Adds a translucent color stop to a gradient pattern.*
- **std::vector**< [ColorStop](#) > [get\\_color\\_stops](#) () const  
*Gets the color stops and offsets for this [Gradient](#).*

## Public Member Functions inherited from Cairo::Pattern

- [Pattern](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Pattern](#) (const [Pattern](#) &)=delete
- [Pattern](#) & [operator=](#) (const [Pattern](#) &)=delete
- virtual [~Pattern](#) ()
- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the pattern's transformation matrix to @matrix.*
- void [get\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the pattern's transformation matrix.*
- [Matrix](#) [get\\_matrix](#) () const  
*Returns the pattern's transformation matrix.*
- [Type](#) [get\\_type](#) () const  
*Returns the type of the pattern.*
- void [set\\_extend](#) ([Extend](#) extend)  
*Sets the mode to be used for drawing outside the area of a pattern.*
- [Extend](#) [get\\_extend](#) () const  
*Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Static Public Member Functions

- static [RefPtr](#)< [LinearGradient](#) > [create](#) (double x0, double y0, double x1, double y1)  
*Create a new linear gradient Cairo::Pattern along the line defined by (x0, y0) and (x1, y1).*

## Protected Member Functions

- [LinearGradient](#) (double x0, double y0, double x1, double y1)

## Protected Member Functions inherited from Cairo::Gradient

- [Gradient](#) ()

## Protected Member Functions inherited from Cairo::Pattern

- [Pattern](#) ()

## Additional Inherited Members

### Public Types inherited from [Cairo::Pattern](#)

- enum class [Type](#) {  
[SOLID](#) = CAIRO\_PATTERN\_TYPE\_SOLID ,  
[SURFACE](#) = CAIRO\_PATTERN\_TYPE\_SURFACE ,  
[LINEAR](#) = CAIRO\_PATTERN\_TYPE\_LINEAR ,  
[RADIAL](#) = CAIRO\_PATTERN\_TYPE\_RADIAL }  
*Type is used to describe the type of a given pattern.*
- enum class [Extend](#) {  
[NONE](#) = CAIRO\_EXTEND\_NONE ,  
[REPEAT](#) = CAIRO\_EXTEND\_REPEAT ,  
[REFLECT](#) = CAIRO\_EXTEND\_REFLECT ,  
[PAD](#) = CAIRO\_EXTEND\_PAD }  
*Cairo::Extend is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).*
- typedef cairo\_pattern\_t [cobject](#)

### Protected Attributes inherited from [Cairo::Pattern](#)

- [cobject](#) \* [m\\_cobject](#)

## 8.12.1 Constructor & Destructor Documentation

### 8.12.1.1 LinearGradient() [1/2]

```
Cairo::LinearGradient::LinearGradient (
    double x0,
    double y0,
    double x1,
    double y1) [protected]
```

### 8.12.1.2 LinearGradient() [2/2]

```
Cairo::LinearGradient::LinearGradient (
    cairo_pattern_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.12.1.3 ~LinearGradient()

```
Cairo::LinearGradient::~~LinearGradient () [override]
```

## 8.12.2 Member Function Documentation

### 8.12.2.1 create()

```
static RefPtr< LinearGradient > Cairo::LinearGradient::create (
    double x0,
    double y0,
    double x1,
    double y1) [static]
```

Create a new linear gradient [Cairo::Pattern](#) along the line defined by (x0, y0) and (x1, y1).

Before using the gradient pattern, a number of color stops should be defined using [Cairo::Gradient::add\\_color\\_stop\\_rgb\(\)](#) or [Cairo::Gradient::add\\_color\\_stop\\_rgba\(\)](#).

Note: The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with [Cairo::Pattern::set\\_matrix\(\)](#).

#### Parameters

<i>x0</i>	x coordinate of the start point
<i>y0</i>	y coordinate of the start point
<i>x1</i>	x coordinate of the end point
<i>y1</i>	y coordinate of the end point

### 8.12.2.2 get\_linear\_points()

```
void Cairo::LinearGradient::get_linear_points (
    double & x0,
    double & y0,
    double & x1,
    double & y1) const
```

#### Parameters

<i>x0</i>	return value for the x coordinate of the first point
<i>y0</i>	return value for the y coordinate of the first point
<i>x1</i>	return value for the x coordinate of the second point
<i>y1</i>	return value for the y coordinate of the second point

Gets the gradient endpoints for a linear gradient.

Since

1.4

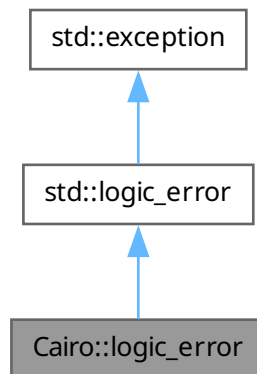
The documentation for this class was generated from the following file:

- [caiomm/pattern.h](#)

## 8.13 Cairo::logic\_error Class Reference

```
#include <cairomm/exception.h>
```

Inheritance diagram for Cairo::logic\_error:



### Public Member Functions

- [logic\\_error](#) (ErrorStatus status)
- [~logic\\_error](#) () noexcept override
- ErrorStatus [get\\_status\\_code](#) () const

### Public Member Functions inherited from std::logic\_error

- [logic\\_error](#) (const **string** &\_\_arg) \_GLIBCXX\_TXN\_SAFE
- virtual const char \* [what](#) () const noexcept

## 8.13.1 Constructor & Destructor Documentation

### 8.13.1.1 logic\_error()

```
Cairo::logic_error::logic_error (  
    ErrorStatus status) [explicit]
```

### 8.13.1.2 ~logic\_error()

```
Cairo::logic_error::~~logic_error () [override], [noexcept]
```



## 8.13.2 Member Function Documentation

### 8.13.2.1 get\_status\_code()

```
ErrorStatus Cairo::logic_error::get_status_code () const
```

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

- cairomm/exception.h

## 8.14 Cairo::Matrix Class Reference

A Transformation matrix.

```
#include <cairomm/matrix.h>
```

Inheritance diagram for Cairo::Matrix:



### Public Member Functions

- [Matrix](#) ()  
*Creates an uninitialized matrix.*
- [Matrix](#) (double xx, double yx, double xy, double yy, double x0, double y0)  
*Creates a matrix Sets to be the affine transformation given by xx, yx, xy, yy, x0, y0.*
- void [translate](#) (double tx, double ty)  
*Applies a translation by tx, ty to the transformation in matrix.*
- void [scale](#) (double sx, double sy)  
*Applies scaling by sx, sy to the transformation in matrix.*
- void [rotate](#) (double radians)  
*Applies rotation by radians to the transformation in matrix.*
- void [invert](#) ()  
*Changes matrix to be the inverse of it's original value.*
- void [multiply](#) ([Matrix](#) &a, [Matrix](#) &b)  
*Multiplies the affine transformations in a and b together and stores the result in this matrix.*
- void [transform\\_distance](#) (double &dx, double &dy) const  
*Transforms the distance vector (dx,dy) by matrix.*
- void [transform\\_point](#) (double &x, double &y) const  
*Transforms the point (x, y) by this matrix.*

## Related Symbols

(Note that these are not member symbols.)

- [Matrix identity\\_matrix](#) ()  
*Returns a [Matrix](#) initialized to the identity matrix.*
- [Matrix translation\\_matrix](#) (double tx, double ty)  
*Returns a [Matrix](#) initialized to a transformation that translates by tx and ty in the X and Y dimensions, respectively.*
- [Matrix scaling\\_matrix](#) (double sx, double sy)  
*Returns a [Matrix](#) initialized to a transformation that scales by sx and sy in the X and Y dimensions, respectively.*
- [Matrix rotation\\_matrix](#) (double radians)  
*Returns a [Matrix](#) initialized to a transformation that rotates by radians.*
- [Matrix operator\\*](#) (const [Matrix](#) &a, const [Matrix](#) &b)  
*Multiplies the affine transformations in a and b together and returns the result.*

### 8.14.1 Detailed Description

A Transformation matrix.

[Cairo::Matrix](#) is used throughout cairomm to convert between different coordinate spaces. A [Matrix](#) holds an affine transformation, such as a scale, rotation, shear, or a combination of these. The transformation of a point (x,y) is given by:

```
x_new = xx * x + xy * y + x0;
y_new = yx * x + yy * y + y0;
```

The current transformation matrix of a [Context](#), represented as a [Matrix](#), defines the transformation from user-space coordinates to device-space coordinates.

See also

[Context::get\\_matrix\(\)](#)

[Context::set\\_matrix\(\)](#)

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 Matrix() [1/2]

```
Cairo::Matrix::Matrix ()
```

Creates an uninitialized matrix.

If you want a matrix initialized to a certain value, either specify the values explicitly with the other constructor or use one of the free functions for initializing matrices with specific scales, rotations, etc.

See also

[identity\\_matrix\(\)](#)

[rotation\\_matrix\(\)](#)

[translation\\_matrix\(\)](#)

[scaling\\_matrix\(\)](#)

### 8.14.2.2 Matrix() [2/2]

```
Cairo::Matrix::Matrix (
    double xx,
    double yx,
    double xy,
    double yy,
    double x0,
    double y0)
```

Creates a matrix Sets to be the affine transformation given by xx, yx, xy, yy, x0, y0.

The transformation is given by:

```
x_new = xx * x + xy * y + x0;
y_new = yx * x + yy * y + y0;
```

#### Parameters

xx	xx component of the affine transformation
yx	yx component of the affine transformation
xy	xy component of the affine transformation
yy	yy component of the affine transformation
x0	X translation component of the affine transformation
y0	Y translation component of the affine transformation

## 8.14.3 Member Function Documentation

### 8.14.3.1 invert()

```
void Cairo::Matrix::invert ()
```

Changes matrix to be the inverse of it's original value.

Not all transformation matrices have inverses; if the matrix collapses points together (it is degenerate), then it has no inverse and this function will throw an exception.

#### Exceptions

--	--

### 8.14.3.2 multiply()

```
void Cairo::Matrix::multiply (
    Matrix & a,
    Matrix & b)
```

Multiplies the affine transformations in a and b together and stores the result in this matrix.

The effect of the resulting transformation is to first apply the transformation in a to the coordinates and then apply the transformation in b to the coordinates.

It is allowable for result to be identical to either a or b.

## Parameters

<i>a</i>	a <a href="#">Matrix</a>
<i>b</i>	a <a href="#">Matrix</a>

## See also

[operator\\*\(\)](#)

**8.14.3.3 rotate()**

```
void Cairo::Matrix::rotate (
    double radians)
```

Applies rotation by radians to the transformation in matrix.

The effect of the new transformation is to first rotate the coordinates by radians, then apply the original transformation to the coordinates.

## Parameters

<i>radians</i>	angle of rotation, in radians. The direction of rotation is defined such that positive angles rotate in the direction from the positive X axis toward the positive Y axis. With the default axis orientation of cairo, positive angles rotate in a clockwise direction.
----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**8.14.3.4 scale()**

```
void Cairo::Matrix::scale (
    double sx,
    double sy)
```

Applies scaling by *sx*, *sy* to the transformation in matrix.

The effect of the new transformation is to first scale the coordinates by *sx* and *sy*, then apply the original transformation to the coordinates.

## Parameters

<i>sx</i>	scale factor in the X direction
<i>sy</i>	scale factor in the Y direction

**8.14.3.5 transform\_distance()**

```
void Cairo::Matrix::transform_distance (
    double & dx,
    double & dy) const
```

Transforms the distance vector (*dx*,*dy*) by matrix.

This is similar to [transform\\_point\(\)](#) except that the translation components of the transformation are ignored. The calculation of the returned vector is as follows:

```
dx2 = dx1 * a + dy1 * c;
dy2 = dx1 * b + dy1 * d;
```

Affine transformations are position invariant, so the same vector always transforms to the same vector. If (*x*<sub>1</sub>,*y*<sub>1</sub>) transforms to (*x*<sub>2</sub>,*y*<sub>2</sub>) then (*x*<sub>1</sub>+*dx*<sub>1</sub>,*y*<sub>1</sub>+*dy*<sub>1</sub>) will transform to (*x*<sub>1</sub>+*dx*<sub>2</sub>,*y*<sub>1</sub>+*dy*<sub>2</sub>) for all values of *x*<sub>1</sub> and *y*<sub>1</sub>.

## Parameters

<i>dx</i>	X component of a distance vector. An in/out parameter
<i>dy</i>	Y component of a distance vector. An in/out parameter

**8.14.3.6 transform\_point()**

```
void Cairo::Matrix::transform_point (
    double & x,
    double & y) const
```

Transforms the point (x, y) by this matrix.

## Parameters

<i>x</i>	X position. An in/out parameter
<i>y</i>	Y position. An in/out parameter

**8.14.3.7 translate()**

```
void Cairo::Matrix::translate (
    double tx,
    double ty)
```

Applies a translation by tx, ty to the transformation in matrix.

The effect of the new transformation is to first translate the coordinates by tx and ty, then apply the original transformation to the coordinates.

## Parameters

<i>tx</i>	amount to translate in the X direction
<i>ty</i>	amount to translate in the Y direction

**8.14.4 Friends And Related Symbol Documentation****8.14.4.1 identity\_matrix()**

```
Matrix identity_matrix () [related]
```

Returns a [Matrix](#) initialized to the identity matrix.

**8.14.4.2 operator\*()**

```
Matrix operator* (
    const Matrix & a,
    const Matrix & b) [related]
```

Multiplies the affine transformations in a and b together and returns the result.

The effect of the resulting transformation is to first apply the transformation in a to the coordinates and then apply the transformation in b to the coordinates.

It is allowable for result to be identical to either a or b.

## Parameters

<i>a</i>	a <a href="#">Matrix</a>
<i>b</i>	a <a href="#">Matrix</a>

**8.14.4.3 rotation\_matrix()**

```
Matrix rotation_matrix (  
    double radians) [related]
```

Returns a [Matrix](#) initialized to a transformation that rotates by radians.

## Parameters

<i>radians</i>	angle of rotation, in radians. The direction of rotation is defined such that positive angles rotate in the direction from the positive X axis toward the positive Y axis. With the default axis orientation of cairo, positive angles rotate in a clockwise direction.
----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**8.14.4.4 scaling\_matrix()**

```
Matrix scaling_matrix (  
    double sx,  
    double sy) [related]
```

Returns a [Matrix](#) initialized to a transformation that scales by sx and sy in the X and Y dimensions, respectively.

## Parameters

<i>sx</i>	scale factor in the X direction
<i>sy</i>	scale factor in the Y direction

**8.14.4.5 translation\_matrix()**

```
Matrix translation_matrix (  
    double tx,  
    double ty) [related]
```

Returns a [Matrix](#) initialized to a transformation that translates by tx and ty in the X and Y dimensions, respectively.

## Parameters

<i>tx</i>	amount to translate in the X direction
<i>ty</i>	amount to translate in the Y direction

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

- cairomm/matrix.h

## 8.15 Cairo::Path Class Reference

A data structure for holding a path.

```
#include <cairomm/path.h>
```

### Public Types

- typedef cairo\_path\_t [cobject](#)

### Public Member Functions

- [Path](#) (cairo\_path\_t \*[cobject](#), bool take\_ownership=false)
- [Path](#) (const [Path](#) &)=delete
- [Path](#) & [operator=](#) (const [Path](#) &)=delete
- virtual [~Path](#) ()
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const

### Protected Attributes

- [cobject](#) \* [m\\_cobject](#)

### 8.15.1 Detailed Description

A data structure for holding a path.

Use [Context::copy\\_path\(\)](#) or [Context::copy\\_path\\_flat\(\)](#) to instantiate a new [Path](#). The application is responsible for freeing the [Path](#) object when it is no longer needed.

### 8.15.2 Member Typedef Documentation

#### 8.15.2.1 cobject

```
typedef cairo_path_t Cairo::Path::cobject
```

### 8.15.3 Constructor & Destructor Documentation

#### 8.15.3.1 Path() [1/2]

```
Cairo::Path::Path (  
    cairo_path_t * cobject,  
    bool take_ownership = false) [explicit]
```

### 8.15.3.2 Path() [2/2]

```
Cairo::Path::Path (
    const Path & ) [delete]
```

### 8.15.3.3 ~Path()

```
virtual Cairo::Path::~~Path () [virtual]
```

## 8.15.4 Member Function Documentation

### 8.15.4.1 cobj() [1/2]

```
cobject * Cairo::Path::cobj () [inline]
```

### 8.15.4.2 cobj() [2/2]

```
const cobject * Cairo::Path::cobj () const [inline]
```

### 8.15.4.3 operator=()

```
Path & Cairo::Path::operator= (
    const Path & ) [delete]
```

## 8.15.5 Member Data Documentation

### 8.15.5.1 m\_cobject

```
cobject* Cairo::Path::m_cobject [protected]
```

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

- cairomm/path.h

## 8.16 Cairo::Pattern Class Reference

[Cairo::Pattern](#) is the paint with which cairo draws.

```
#include <cairomm/pattern.h>
```

Inheritance diagram for Cairo::Pattern:





## Public Types

- enum class [Type](#) {  
[SOLID](#) = CAIRO\_PATTERN\_TYPE\_SOLID ,  
[SURFACE](#) = CAIRO\_PATTERN\_TYPE\_SURFACE ,  
[LINEAR](#) = CAIRO\_PATTERN\_TYPE\_LINEAR ,  
[RADIAL](#) = CAIRO\_PATTERN\_TYPE\_RADIAL }

*Type is used to describe the type of a given pattern.*

- enum class [Extend](#) {  
[NONE](#) = CAIRO\_EXTEND\_NONE ,  
[REPEAT](#) = CAIRO\_EXTEND\_REPEAT ,  
[REFLECT](#) = CAIRO\_EXTEND\_REFLECT ,  
[PAD](#) = CAIRO\_EXTEND\_PAD }

*Cairo::Extend is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).*

- typedef cairo\_pattern\_t [cobject](#)

## Public Member Functions

- [Pattern](#) (cairo\_pattern\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Pattern](#) (const [Pattern](#) &)=delete
- [Pattern](#) & [operator=](#) (const [Pattern](#) &)=delete
- virtual ~[Pattern](#) ()
- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the pattern's transformation matrix to @matrix.*
- void [get\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the pattern's transformation matrix.*
- [Matrix](#) [get\\_matrix](#) () const  
*Returns the pattern's transformation matrix.*
- [Type](#) [get\\_type](#) () const  
*Returns the type of the pattern.*
- void [set\\_extend](#) ([Extend](#) extend)  
*Sets the mode to be used for drawing outside the area of a pattern.*
- [Extend](#) [get\\_extend](#) () const  
*Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Protected Member Functions

- [Pattern](#) ()

## Protected Attributes

- [cobject](#) \* [m\\_cobject](#)

### 8.16.1 Detailed Description

[Cairo::Pattern](#) is the paint with which cairo draws.

The primary use of patterns is as the source for all cairo drawing operations, although they can also be used as masks, that is, as the brush too.

This is a reference-counted object that should be used via [Cairo::RefPtr](#).

### 8.16.2 Member Typedef Documentation

#### 8.16.2.1 cobject

```
typedef cairo_pattern_t Cairo::Pattern::cobject
```

### 8.16.3 Member Enumeration Documentation

#### 8.16.3.1 Extend

```
enum class Cairo::Pattern::Extend [strong]
```

[Cairo::Extend](#) is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).

Mesh patterns are not affected by the extend mode.

The default extend mode is [Cairo::Pattern::Extend::NONE](#) for surface patterns and [Cairo::Pattern::Extend::PAD](#) for gradient patterns.

New entries may be added in future versions.

##### Enumerator

NONE	Pixels outside of the source pattern are fully transparent.
REPEAT	The pattern is tiled by repeating.
REFLECT	The pattern is tiled by reflecting at the edges (Implemented for surface patterns since 1.6)
PAD	Pixels outside of the pattern copy the closest pixel from the source (Since 1.2; but only implemented for surface patterns since 1.6)

#### 8.16.3.2 Type

```
enum class Cairo::Pattern::Type [strong]
```

Type is used to describe the type of a given pattern.

The pattern type can be queried with [Pattern::get\\_type\(\)](#).

New entries may be added in future versions.

##### Since

1.2

## Enumerator

SOLID	The pattern is a solid (uniform) color. It may be opaque or translucent.
SURFACE	The pattern is based on a surface (an image)
LINEAR	The pattern is a linear gradient.
RADIAL	The pattern is a radial gradient.

## 8.16.4 Constructor & Destructor Documentation

### 8.16.4.1 Pattern() [1/3]

```
Cairo::Pattern::Pattern (
    cairo_pattern_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.16.4.2 Pattern() [2/3]

```
Cairo::Pattern::Pattern (
    const Pattern & ) [delete]
```

### 8.16.4.3 ~Pattern()

```
virtual Cairo::Pattern::~~Pattern () [virtual]
```

### 8.16.4.4 Pattern() [3/3]

```
Cairo::Pattern::Pattern () [protected]
```

## 8.16.5 Member Function Documentation

### 8.16.5.1 cobj() [1/2]

```
cobject * Cairo::Pattern::cobj () [inline]
```

#### 8.16.5.2 cobj() [2/2]

```
const cobject * Cairo::Pattern::cobj () const [inline]
```

#### 8.16.5.3 get\_extend()

```
Extend Cairo::Pattern::get_extend () const
```

Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.

Since

1.12

#### 8.16.5.4 get\_matrix() [1/2]

```
Matrix Cairo::Pattern::get_matrix () const
```

Returns the pattern's transformation matrix.

Since

1.8

#### 8.16.5.5 get\_matrix() [2/2]

```
void Cairo::Pattern::get_matrix (
    Matrix & matrix) const
```

Returns the pattern's transformation matrix.

#### 8.16.5.6 get\_type()

```
Type Cairo::Pattern::get_type () const
```

Returns the type of the pattern.

Since

1.2

#### 8.16.5.7 operator=()

```
Pattern & Cairo::Pattern::operator= (
    const Pattern & ) [delete]
```

### 8.16.5.8 reference()

```
void Cairo::Pattern::reference () const
```

### 8.16.5.9 set\_extend()

```
void Cairo::Pattern::set_extend (
    Extend extend)
```

Sets the mode to be used for drawing outside the area of a pattern.

See Cairo::Extend for details on the semantics of each extend strategy.

The default extend mode is [Cairo::Pattern::Extend::NONE](#) for surface patterns and [Cairo::Pattern::Extend::PAD](#) for gradient patterns.

## Parameters

<i>Cairo::Extend</i>	describing how the area outside of the pattern will be drawn
----------------------	--------------------------------------------------------------

## Since

1.12

**8.16.5.10 set\_matrix()**

```
void Cairo::Pattern::set_matrix (
    const Matrix & matrix)
```

Sets the pattern's transformation matrix to @matrix.

This matrix is a transformation from user space to pattern space.

When a pattern is first created it always has the identity matrix for its transformation matrix, which means that pattern space is initially identical to user space.

Important: Please note that the direction of this transformation matrix is from user space to pattern space. This means that if you imagine the flow from a pattern to user space (and on to device space), then coordinates in that flow will be transformed by the inverse of the pattern matrix.

For example, if you want to make a pattern appear twice as large as it does by default the correct code to use is:  
`pattern->set_matrix(scaling_matrix(0.5, 0.5));`

Meanwhile, using values of 2.0 rather than 0.5 in the code above would cause the pattern to appear at half of its default size.

Also, please note the discussion of the user-space locking semantics of `set_source()`.

**8.16.5.11 unreference()**

```
void Cairo::Pattern::unreference () const
```

**8.16.6 Member Data Documentation****8.16.6.1 m\_cobject**

```
cobject* Cairo::Pattern::m_cobject [protected]
```

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

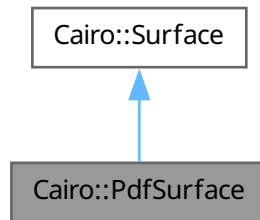
- `caiomm/pattern.h`

## 8.17 Cairo::PdfSurface Class Reference

A PdfSurface provides a way to render PDF documents from cairo.

```
#include <cairomm/surface.h>
```

Inheritance diagram for Cairo::PdfSurface:



### Public Member Functions

- [PdfSurface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~PdfSurface](#) () override
- void [set\\_size](#) (double width\_in\_points, double height\_in\_points)  
*Changes the size of a PDF surface for the current (and subsequent) pages.*
- void [restrict\\_to\\_version](#) (PdfVersion version)  
*Restricts the generated PDF file to version.*

### Public Member Functions inherited from [Cairo::Surface](#)

- [Surface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & operator= (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* **data**, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()

Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.

- void [mark\\_dirty](#) ()  
Tells cairo to consider the data buffer dirty.
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
Marks a rectangular area of the given surface dirty.
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
Returns a previous device offset set by [set\\_device\\_offset\(\)](#).
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.
- void [set\\_device\\_scale](#) (double scale)  
Sets x and y scale to the same value.
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
Returns a previous device scale set by [set\\_device\\_scale\(\)](#).
- double [get\\_device\\_scale](#) () const  
Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
Set the horizontal and vertical resolution for image fallbacks.
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.
- [Type get\\_type](#) () const
- [Content get\\_content](#) () const  
This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.
- void [copy\\_page](#) ()  
Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.
- void [show\\_page](#) ()  
Emits and clears the current page for backends that support multiple pages.
- bool [has\\_show\\_text\\_glyphs](#) () const  
Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.
- void [write\\_to\\_png](#) (const **std::string** &filename)  
Writes the contents of surface to a new file filename as a PNG image.
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)  
Writes the [Surface](#) to the write function.
- [RefPtr< Device > get\\_device](#) ()  
This function returns the device for a surface.
- [cobject \\* cobj](#) ()  
Provides acces to the underlying C cairo surface.
- const [cobject \\* cobj](#) () const  
Provides acces to the underlying C cairo surface.



### Static Public Member Functions

- static [RefPtr](#)< [PdfSurface](#) > [create](#) ( [std::string](#) filename, double width\_in\_points, double height\_in\_points)  
*Creates a [PdfSurface](#) with a specified dimensions that will be saved as the given filename.*
- static [RefPtr](#)< [PdfSurface](#) > [create\\_for\\_stream](#) (const [SlotWriteFunc](#) &write\_func, double width\_in\_points, double height\_in\_points)  
*Creates a [PdfSurface](#) with a specified dimensions that will be written to the given write function instead of saved directly to disk.*
- static const [std::vector](#)< [PdfVersion](#) > [get\\_versions](#) ()  
*Retrieves the list of PDF versions supported by cairo.*
- static [std::string](#) [version\\_to\\_string](#) ([PdfVersion](#) version)  
*Get the string representation of the given version id.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > other, [Content](#) content, int width, int height)  
*Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > &target, double x, double y, double width, double height)  
*Create a new surface that is a rectangle within the target surface.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {  
[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,  
[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,  
[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,  
[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,  
[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,  
[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,  
[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,  
[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,  
[WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,  
[QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,  
[SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,  
[QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,  
[RECORDING](#) = CAIRO\_SURFACE\_TYPE\_RECORDING ,  
[VG](#) = CAIRO\_SURFACE\_TYPE\_VG ,  
[GL](#) = CAIRO\_SURFACE\_TYPE\_GL ,  
[DRM](#) = CAIRO\_SURFACE\_TYPE\_DRM ,  
[TEE](#) = CAIRO\_SURFACE\_TYPE\_TEE ,  
[XML](#) = CAIRO\_SURFACE\_TYPE\_XML ,  
[SKIA](#) = CAIRO\_SURFACE\_TYPE\_SKIA ,  
[SUBSURFACE](#) = CAIRO\_SURFACE\_TYPE\_SUBSURFACE }

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }  
*Format is used to identify the memory format of image data.*
- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)  
*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*
- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)  
*This is the type of function which is called when a backend needs to read data from an input stream.*
- typedef sigc::slot< void()> [SlotDestroy](#)  
*For instance, void on\_destroy();.*
- typedef cairo\_surface\_t [cobject](#)  
*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)  
*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.17.1 Detailed Description

A PdfSurface provides a way to render PDF documents from cairo.

This surface is not rendered to the screen but instead renders the drawing to a PDF file on disk.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with PDF support

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 PdfSurface()

```
Cairo::PdfSurface::PdfSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.17.2.2 ~PdfSurface()

```
Cairo::PdfSurface::~~PdfSurface () [override]
```

## 8.17.3 Member Function Documentation

### 8.17.3.1 create()

```
static RefPtr< PdfSurface > Cairo::PdfSurface::create (
    std::string filename,
    double width_in_points,
    double height_in_points) [static]
```

Creates a [PdfSurface](#) with a specified dimensions that will be saved as the given filename.

#### Parameters

<i>filename</i>	The name of the PDF file to save the surface to
<i>width_in_points</i>	The width of the PDF document in points
<i>height_in_points</i>	The height of the PDF document in points

#### Since

1.2

#### Examples

[pdf-surface.cc](#).

### 8.17.3.2 create\_for\_stream()

```
static RefPtr< PdfSurface > Cairo::PdfSurface::create_for_stream (
    const SlotWriteFunc & write_func,
    double width_in_points,
    double height_in_points) [static]
```

Creates a [PdfSurface](#) with a specified dimensions that will be written to the given write function instead of saved directly to disk.

#### Parameters

<i>write_func</i>	The function to be called when the backend needs to write data to an output stream
<i>width_in_points</i>	The width of the PDF document in points
<i>height_in_points</i>	The height of the PDF document in points

#### Since

1.8

### 8.17.3.3 `get_versions()`

```
static const std::vector< PdfVersion > Cairo::PdfSurface::get_versions () [static]
```

Retrieves the list of PDF versions supported by cairo.

See [restrict\\_to\\_version\(\)](#).

Since

1.10

### 8.17.3.4 `restrict_to_version()`

```
void Cairo::PdfSurface::restrict_to_version (  
    PdfVersion version)
```

Restricts the generated PDF file to version.

See [get\\_versions\(\)](#) for a list of available version values that can be used here.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

Since

1.10

### 8.17.3.5 `set_size()`

```
void Cairo::PdfSurface::set_size (  
    double width_in_points,  
    double height_in_points)
```

Changes the size of a PDF surface for the current (and subsequent) pages.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface or immediately after completing a page with either [Context::show\\_page\(\)](#) or [Context::copy\\_page\(\)](#).

Parameters

<i>width_in_points</i>	new surface width, in points (1 point == 1/72.0 inch)
<i>height_in_points</i>	new surface height, in points (1 point == 1/72.0 inch)

### 8.17.3.6 version\_to\_string()

```
static std::string Cairo::PdfSurface::version_to_string (
    PdfVersion version) [static]
```

Get the string representation of the given version id.

This function will return an empty string if version isn't valid. See [get\\_versions\(\)](#) for a way to get the list of valid version ids.

Since

1.10

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

- [caiomm/surface.h](#)

## 8.18 Cairo::PsSurface Class Reference

A PsSurface provides a way to render PostScript documents from cairo.

```
#include <caiomm/surface.h>
```

Inheritance diagram for Cairo::PsSurface:



### Public Member Functions

- [PsSurface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~PsSurface](#) () override
- void [set\\_size](#) (double width\_in\_points, double height\_in\_points)  
*Changes the size of a PostScript surface for the current (and subsequent) pages.*
- void [dsc\\_comment](#) ( **std::string** comment)  
*Emit a comment into the PostScript output for the given surface.*
- void [dsc\\_begin\\_setup](#) ()

This function indicates that subsequent calls to `dsc_comment()` should direct comments to the Setup section of the PostScript output.

- void `dsc_begin_page_setup` ()

This function indicates that subsequent calls to `dsc_comment()` should direct comments to the PageSetup section of the PostScript output.

- void `set_eps` (bool eps)

If eps is true, the PostScript surface will output Encapsulated PostScript.

- bool `get_eps` () const

Check whether the PostScript surface will output Encapsulated PostScript.

- void `restrict_to_level` (PsLevel level)

Restricts the generated PostScript file to @level.

## Public Member Functions inherited from `Cairo::Surface`

- `Surface` (cairo\_surface\_t \*cobject, bool has\_reference=false)

Create a C++ wrapper for the C instance.

- `Surface` (const `Surface` &)=delete

- `Surface & operator=` (const `Surface` &)=delete

- virtual `~Surface` ()

- const unsigned char \* `get_mime_data` (const `std::string` &mime\_type, unsigned long &length)

Return mime data previously attached to surface using the specified mime type.

- void `set_mime_data` (const `std::string` &mime\_type, unsigned char \* data, unsigned long length, const `SlotDestroy` &slot\_destroy)

Attach an image in the format mime\_type to surface.

- void `unset_mime_data` (const `std::string` &mime\_type)

Remove the data from a surface.

- void `get_font_options` (`FontOptions` &options) const

Retrieves the default font rendering options for the surface.

- void `finish` ()

This function finishes the surface and drops all references to external resources.

- void `flush` ()

Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.

- void `mark_dirty` ()

Tells cairo to consider the data buffer dirty.

- void `mark_dirty` (int x, int y, int width, int height)

Marks a rectangular area of the given surface dirty.

- void `set_device_offset` (double x\_offset, double y\_offset)

Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.

- void `get_device_offset` (double &x\_offset, double &y\_offset) const

Returns a previous device offset set by `set_device_offset()`.

- void `set_device_scale` (double x\_scale, double y\_scale)

Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.

- void `set_device_scale` (double scale)

Sets x and y scale to the same value.

- void `get_device_scale` (double &x\_scale, double &y\_scale) const

Returns a previous device scale set by `set_device_scale()`.

- double `get_device_scale` () const

Returns the x and y average of a previous device scale set by `set_device_scale()`.

- void `set_fallback_resolution` (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)

Set the horizontal and vertical resolution for image fallbacks.

- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type get\\_type](#) () const
- [Content get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const  
*Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)  
*Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)  
*Writes the [Surface](#) to the write function.*
- [RefPtr< Device > get\\_device](#) ()  
*This function returns the device for a surface.*
- [cobject \\* cobj](#) ()  
*Provides acces to the underlying C cairo surface.*
- const [cobject \\* cobj](#) () const  
*Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr< PsSurface > create](#) ( **std::string** filename, double width\_in\_points, double height\_in\_points)  
*Creates a [PsSurface](#) with a specified dimensions that will be saved as the given filename.*
- static [RefPtr< PsSurface > create\\_for\\_stream](#) (const [SlotWriteFunc](#) &write\_func, double width\_in\_points, double height\_in\_points)  
*Creates a [PsSurface](#) with a specified dimensions that will be written to the given write function instead of saved directly to disk.*
- static const **std::vector< PsLevel >** [get\\_levels](#) ()  
*Used to retrieve the list of supported levels.*
- static **std::string** [level\\_to\\_string](#) ([PsLevel](#) level)  
*Get the string representation of the given level id.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr< Surface > create](#) (const [RefPtr< Surface >](#) other, [Content](#) content, int width, int height)  
*Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr< Surface > create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)  
*Create a new surface that is a rectangle within the target surface.*

## Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {  
[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,  
[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,  
[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,  
[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,  
[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,  
[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,  
[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,  
[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,  
[WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,  
[QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,  
[SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,  
[QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,  
[RECORDING](#) = CAIRO\_SURFACE\_TYPE\_RECORDING ,  
[VG](#) = CAIRO\_SURFACE\_TYPE\_VG ,  
[GL](#) = CAIRO\_SURFACE\_TYPE\_GL ,  
[DRM](#) = CAIRO\_SURFACE\_TYPE\_DRM ,  
[TEE](#) = CAIRO\_SURFACE\_TYPE\_TEE ,  
[XML](#) = CAIRO\_SURFACE\_TYPE\_XML ,  
[SKIA](#) = CAIRO\_SURFACE\_TYPE\_SKIA ,  
[SUBSURFACE](#) = CAIRO\_SURFACE\_TYPE\_SUBSURFACE }  
*Cairo::Surface::Type is used to describe the type of a given surface.*
- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }  
*Format is used to identify the memory format of image data.*
- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)  
*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*
- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)  
*This is the type of function which is called when a backend needs to read data from an input stream.*
- typedef sigc::slot< void()> [SlotDestroy](#)  
*For instance, `void on_destroy();`.*
- typedef cairo\_surface\_t [cobject](#)  
*The underlying C cairo surface type.*

### Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)  
*The underlying C cairo surface type that is wrapped by this [Surface](#).*



### 8.18.1 Detailed Description

A PsSurface provides a way to render PostScript documents from cairo.

This surface is not rendered to the screen but instead renders the drawing to a PostScript file on disk.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with PostScript support

### 8.18.2 Constructor & Destructor Documentation

#### 8.18.2.1 PsSurface()

```
Cairo::PsSurface::PsSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

#### 8.18.2.2 ~PsSurface()

```
Cairo::PsSurface::~~PsSurface () [override]
```

### 8.18.3 Member Function Documentation

#### 8.18.3.1 create()

```
static RefPtr< PsSurface > Cairo::PsSurface::create (
    std::string filename,
    double width_in_points,
    double height_in_points) [static]
```

Creates a [PsSurface](#) with a specified dimensions that will be saved as the given filename.

#### Parameters

<i>filename</i>	The name of the PostScript file to save the surface to
<i>width_in_points</i>	The width of the PostScript document in points
<i>height_in_points</i>	The height of the PostScript document in points

#### Examples

[ps-surface.cc](#).

### 8.18.3.2 create\_for\_stream()

```
static RefPtr< PsSurface > Cairo::PsSurface::create_for_stream (
    const SlotWriteFunc & write_func,
    double width_in_points,
    double height_in_points) [static]
```

Creates a [PsSurface](#) with a specified dimensions that will be written to the given write function instead of saved directly to disk.

#### Parameters

<i>write_func</i>	The function to be called when the backend needs to write data to an output stream
<i>width_in_points</i>	The width of the PostScript document in points
<i>height_in_points</i>	The height of the PostScript document in points

Since

1.8

### 8.18.3.3 dsc\_begin\_page\_setup()

```
void Cairo::PsSurface::dsc_begin_page_setup ()
```

This function indicates that subsequent calls to [dsc\\_comment\(\)](#) should direct comments to the PageSetup section of the PostScript output.

This function call is only needed for the first page of a surface. It should be called after any call to [dsc\\_begin\\_setup\(\)](#) and before any drawing is performed to the surface.

### 8.18.3.4 dsc\_begin\_setup()

```
void Cairo::PsSurface::dsc_begin_setup ()
```

This function indicates that subsequent calls to [dsc\\_comment\(\)](#) should direct comments to the Setup section of the PostScript output.

This function should be called at most once per surface, and must be called before any call to [dsc\\_begin\\_page\\_setup\(\)](#) and before any drawing is performed to the surface.

### 8.18.3.5 dsc\_comment()

```
void Cairo::PsSurface::dsc_comment (
    std::string comment)
```

Emit a comment into the PostScript output for the given surface.

See the cairo reference documentation for more information.

## Parameters

<i>comment</i>	a comment string to be emitted into the PostScript output
----------------	-----------------------------------------------------------

**8.18.3.6 get\_eps()**

```
bool Cairo::PsSurface::get_eps () const
```

Check whether the PostScript surface will output Encapsulated PostScript.

Since

1.8

**8.18.3.7 get\_levels()**

```
static const std::vector< PsLevel > Cairo::PsSurface::get_levels () [static]
```

Used to retrieve the list of supported levels.

See [restrict\\_to\\_level\(\)](#).

Since

1.6

**8.18.3.8 level\_to\_string()**

```
static std::string Cairo::PsSurface::level_to_string (
    PsLevel level) [static]
```

Get the string representation of the given level id.

This function will return an empty string if level id isn't valid. See [get\\_levels\(\)](#) for a way to get the list of valid level ids.

Returns

the string associated to given level.

## Parameters

<i>level</i>	a level id
--------------	------------

Since

1.6

**8.18.3.9 restrict\_to\_level()**

```
void Cairo::PsSurface::restrict_to_level (
    PsLevel level)
```

Restricts the generated PostScript file to @level.

See [get\\_levels\(\)](#) for a list of available level values that can be used here.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

## Parameters

<i>level</i>	PostScript level
--------------	------------------

## Since

1.6

**8.18.3.10 set\_eps()**

```
void Cairo::PsSurface::set_eps (
    bool eps)
```

If eps is true, the PostScript surface will output Encapsulated PostScript.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface. An Encapsulated Postscript file should never contain more than one page.

## Since

1.6

**8.18.3.11 set\_size()**

```
void Cairo::PsSurface::set_size (
    double width_in_points,
    double height_in_points)
```

Changes the size of a PostScript surface for the current (and subsequent) pages.

This function should only be called before any drawing operations have been performed on the current page. The simplest way to do this is to call this function immediately after creating the surface or immediately after completing a page with either [Context::show\\_page\(\)](#) or [Context::copy\\_page\(\)](#).

## Parameters

<i>width_in_points</i>	new surface width, in points (1 point == 1/72.0 inch)
<i>height_in_points</i>	new surface height, in points (1 point == 1/72.0 inch)

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

- cairomm/surface.h

## 8.19 Cairo::QuartzFontFace Class Reference

The Quartz font backend is primarily used to render text on Apple MacOS X systems.

```
#include <cairomm/quartz_font.h>
```

Inheritance diagram for Cairo::QuartzFontFace:



### Static Public Member Functions

- static [RefPtr< QuartzFontFace > create](#) (CGFontRef font)  
*Creates a new font for the Quartz font backend based on a CGFontRef.*
- static [RefPtr< QuartzFontFace > create](#) (ATSUFontID font\_id)  
*Creates a new font for the Quartz font backend based on an ATSUFontID.*

### Protected Member Functions

- [QuartzFontFace](#) (CGFontRef font)
- [QuartzFontFace](#) (ATSUFontID font\_id)

### Additional Inherited Members

### Public Types inherited from Cairo::FontFace

- typedef cairo\_font\_face\_t [cobject](#)

### Public Member Functions inherited from Cairo::FontFace

- [FontFace](#) (cairo\_font\_face\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [FontFace](#) (const [FontFace](#) &)=delete
- [FontFace](#) & [operator=](#) (const [FontFace](#) &)=delete
- virtual [~FontFace](#) ()
- [FontType](#) [get\\_type](#) () const  
*Returns the type of the backend used to create a font face.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Protected Attributes inherited from [Cairo::FontFace](#)

- [cobject](#) \* [m\\_cobject](#)

### 8.19.1 Detailed Description

The Quartz font backend is primarily used to render text on Apple MacOS X systems.

The CGFont API is used for the internal implementation of the font backend methods.

Since

1.8

### 8.19.2 Constructor & Destructor Documentation

#### 8.19.2.1 QuartzFontFace() [1/2]

```
Cairo::QuartzFontFace::QuartzFontFace (
    CGFontRef font) [protected]
```

#### 8.19.2.2 QuartzFontFace() [2/2]

```
Cairo::QuartzFontFace::QuartzFontFace (
    ATSUIFontID font_id) [protected]
```

### 8.19.3 Member Function Documentation

#### 8.19.3.1 create() [1/2]

```
static RefPtr< QuartzFontFace > Cairo::QuartzFontFace::create (
    ATSUIFontID font_id) [static]
```

Creates a new font for the Quartz font backend based on an ATSUIFontID.

This font can then be used with [Context::set\\_font\\_face\(\)](#) or [ScaledFont::create\(\)](#).

Parameters

<i>font_id</i>	an ATSUIFontID for the font.
----------------	------------------------------

Since

1.8

#### 8.19.3.2 create() [2/2]

```
static RefPtr< QuartzFontFace > Cairo::QuartzFontFace::create (
    CGFontRef font) [static]
```

Creates a new font for the Quartz font backend based on a CGFontRef.

This font can then be used with [Context::set\\_font\\_face\(\)](#) or [ScaledFont::create\(\)](#).

## Parameters

<i>font</i>	a CGFontRef obtained through a method external to cairo.
-------------	----------------------------------------------------------

## Since

1.8

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

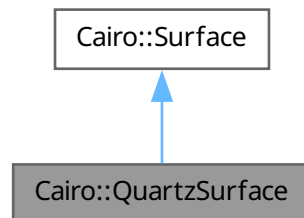
- cairomm/quartz\_font.h

## 8.20 Cairo::QuartzSurface Class Reference

A [QuartzSurface](#) provides a way to render within Apple Mac OS X.

```
#include <cairomm/quartz_surface.h>
```

Inheritance diagram for Cairo::QuartzSurface:



### Public Member Functions

- [QuartzSurface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~QuartzSurface](#) () override
- CGContextRef [get\\_cg\\_context](#) () const  
*Returns the CGContextRef associated with this surface, or NULL if none.*

## Public Member Functions inherited from Cairo::Surface

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & [operator=](#) (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* data, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const



- Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)
  - Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)
  - Writes the [Surface](#) to the write function.*
- [RefPtr< Device >](#) [get\\_device](#) ()
  - This function returns the device for a surface.*
- [cobject \\* cobj](#) ()
  - Provides acces to the underlying C cairo surface.*
- const [cobject \\* cobj](#) () const
  - Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr< QuartzSurface >](#) [create](#) (CGContextRef cg\_context, int width, int height)
  - Creates a Quartz surface that wraps the given CGContext.*
- static [RefPtr< QuartzSurface >](#) [create](#) ([Format](#) format, int width, int height)
  - Creates a Quartz surface backed by a CGBitmap.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) other, [Content](#) content, int width, int height)
  - Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)
  - Create a new surface that is a rectangle within the target surface.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {
  - [IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,
  - [PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,
  - [PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,
  - [XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,
  - [XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,
  - [GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,
  - [QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,
  - [WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,
  - [WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,
  - [BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,
  - [DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,
  - [SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,
  - [OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,
  - [WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,
  - [QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,
  - [SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,
  - [QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,
  - [RECORDING](#) = CAIRO\_SURFACE\_TYPE\_RECORDING ,
  - [VG](#) = CAIRO\_SURFACE\_TYPE\_VG ,

```

GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)

*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*

- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)

*This is the type of function which is called when a backend needs to read data from an input stream.*

- typedef sigc::slot< void()> [SlotDestroy](#)

*For instance, void on\_destroy();.*

- typedef cairo\_surface\_t [cobject](#)

*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)

*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.20.1 Detailed Description

A [QuartzSurface](#) provides a way to render within Apple Mac OS X.

If you want to draw to the screen within a Mac OS X application, you should use this [Surface](#) type.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with (native) Quartz support (requires [Cairo](#) > 1.4.0)

#### Since

1.4

### 8.20.2 Constructor & Destructor Documentation

#### 8.20.2.1 [QuartzSurface\(\)](#)

```

Cairo::QuartzSurface::QuartzSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]

```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

## Since

1.4

## 8.20.2.2 ~QuartzSurface()

```
Cairo::QuartzSurface::~QuartzSurface () [override]
```

## 8.20.3 Member Function Documentation

## 8.20.3.1 create() [1/2]

```
static RefPtr< QuartzSurface > Cairo::QuartzSurface::create (
    CGContextRef cg_context,
    int width,
    int height) [static]
```

Creates a Quartz surface that wraps the given CGContext.

The CGContext is assumed to be in the standard [Cairo](#) coordinate space (that is, with the origin at the upper left and the Y axis increasing downward). If the CGContext is in the Quartz coordinate space (with the origin at the bottom left), then it should be flipped before this function is called. The flip can be accomplished using a translate and a scale; for example:

```
CGContextTranslateCTM (cgContext, 0.0, height);
CGContextScaleCTM (cgContext, 1.0, -1.0);
```

All [Cairo](#) operations are implemented in terms of Quartz operations, as long as Quartz-compatible elements are used (such as Quartz fonts).

## Parameters

<i>cg_context</i>	the CGContext to create a surface for
-------------------	---------------------------------------

## Returns

the newly created surface

## Since

1.4

## 8.20.3.2 create() [2/2]

```
static RefPtr< QuartzSurface > Cairo::QuartzSurface::create (
    Format format,
    int width,
    int height) [static]
```

Creates a Quartz surface backed by a CGBitmap.

The surface is created using the [Device](#) RGB (or [Device](#) Gray, for A8) color space. All [Cairo](#) operations, including those that require software rendering, will succeed on this surface.

## Parameters

<i>format</i>	format of pixels in the surface to create
<i>width</i>	width of the surface, in pixels
<i>height</i>	height of the surface, in pixels

## Returns

the newly created surface

## Since

1.4

### 8.20.3.3 get\_cg\_context()

```
CGContextRef Cairo::QuartzSurface::get_cg_context () const
```

Returns the CGContextRef associated with this surface, or NULL if none.

Also returns NULL if the surface is not a Quartz surface.

## Returns

CGContextRef or NULL if no CGContextRef available.

## Since

1.4

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

- cairomm/quartz\_surface.h

## 8.21 Cairo::RadialGradient Class Reference

```
#include <cairomm/pattern.h>
```

Inheritance diagram for Cairo::RadialGradient:



## Public Member Functions

- [RadialGradient](#) (cairo\_pattern\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- void [get\\_radial\\_circles](#) (double &x0, double &y0, double &r0, double &x1, double &y1, double &r1) const  
*Gets the gradient endpoint circles for a radial gradient, each specified as a center coordinate and a radius.*
- [~RadialGradient](#) () override

## Public Member Functions inherited from Cairo::Gradient

- [Gradient](#) (cairo\_pattern\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~Gradient](#) () override
- void [add\\_color\\_stop\\_rgb](#) (double offset, double red, double green, double blue)  
*Adds an opaque color stop to a gradient pattern.*
- void [add\\_color\\_stop\\_rgba](#) (double offset, double red, double green, double blue, double alpha)  
*Adds a translucent color stop to a gradient pattern.*
- **std::vector**< [ColorStop](#) > [get\\_color\\_stops](#) () const  
*Gets the color stops and offsets for this [Gradient](#).*

## Public Member Functions inherited from Cairo::Pattern

- [Pattern](#) (cairo\_pattern\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Pattern](#) (const [Pattern](#) &)=delete
- [Pattern](#) & [operator=](#) (const [Pattern](#) &)=delete
- virtual [~Pattern](#) ()
- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the pattern's transformation matrix to @matrix.*
- void [get\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the pattern's transformation matrix.*
- [Matrix](#) [get\\_matrix](#) () const  
*Returns the pattern's transformation matrix.*
- [Type](#) [get\\_type](#) () const  
*Returns the type of the pattern.*
- void [set\\_extend](#) ([Extend](#) extend)  
*Sets the mode to be used for drawing outside the area of a pattern.*
- [Extend](#) [get\\_extend](#) () const  
*Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Static Public Member Functions

- static **RefPtr**< [RadialGradient](#) > [create](#) (double cx0, double cy0, double radius0, double cx1, double cy1, double radius1)  
*Creates a new radial gradient #cairo\_pattern\_t between the two circles defined by (cx0, cy0, radius0) and (cx1, cy1, radius1).*

**Protected Member Functions**

- [RadialGradient](#) (double cx0, double cy0, double radius0, double cx1, double cy1, double radius1)

**Protected Member Functions inherited from [Cairo::Gradient](#)**

- [Gradient](#) ()

**Protected Member Functions inherited from [Cairo::Pattern](#)**

- [Pattern](#) ()

**Additional Inherited Members****Public Types inherited from [Cairo::Pattern](#)**

- enum class [Type](#) {  
[SOLID](#) = CAIRO\_PATTERN\_TYPE\_SOLID ,  
[SURFACE](#) = CAIRO\_PATTERN\_TYPE\_SURFACE ,  
[LINEAR](#) = CAIRO\_PATTERN\_TYPE\_LINEAR ,  
[RADIAL](#) = CAIRO\_PATTERN\_TYPE\_RADIAL }  
*Type is used to describe the type of a given pattern.*
- enum class [Extend](#) {  
[NONE](#) = CAIRO\_EXTEND\_NONE ,  
[REPEAT](#) = CAIRO\_EXTEND\_REPEAT ,  
[REFLECT](#) = CAIRO\_EXTEND\_REFLECT ,  
[PAD](#) = CAIRO\_EXTEND\_PAD }  
*Cairo::Extend is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).*
- typedef cairo\_pattern\_t [cobject](#)

**Protected Attributes inherited from [Cairo::Pattern](#)**

- [cobject](#) \* [m\\_cobject](#)

**8.21.1 Constructor & Destructor Documentation****8.21.1.1 RadialGradient() [1/2]**

```
Cairo::RadialGradient::RadialGradient (
    double cx0,
    double cy0,
    double radius0,
    double cx1,
    double cy1,
    double radius1) [protected]
```

**8.21.1.2 RadialGradient() [2/2]**

```
Cairo::RadialGradient::RadialGradient (
    cairo_pattern_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

## 8.21.1.3 ~RadialGradient()

```
Cairo::RadialGradient::~~RadialGradient () [override]
```

## 8.21.2 Member Function Documentation

## 8.21.2.1 create()

```
static RefPtr< RadialGradient > Cairo::RadialGradient::create (
    double cx0,
    double cy0,
    double radius0,
    double cx1,
    double cy1,
    double radius1) [static]
```

Creates a new radial gradient #cairo\_pattern\_t between the two circles defined by (cx0, cy0, radius0) and (cx1, cy1, radius1).

Before using the gradient pattern, a number of color stops should be defined using [Cairo::Gradient::add\\_color\\_stop\\_rgb\(\)](#) or [Cairo::Gradient::add\\_color\\_stop\\_rgba\(\)](#).

## Note

The coordinates here are in pattern space. For a new pattern, pattern space is identical to user space, but the relationship between the spaces can be changed with [Cairo::Pattern::set\\_matrix\(\)](#).

## Parameters

<i>cx0</i>	x coordinate for the center of the start circle
<i>cy0</i>	y coordinate for the center of the start circle
<i>radius0</i>	radius of the start circle
<i>cx1</i>	x coordinate for the center of the end circle
<i>cy1</i>	y coordinate for the center of the end circle
<i>radius1</i>	radius of the end circle

## 8.21.2.2 get\_radial\_circles()

```
void Cairo::RadialGradient::get_radial_circles (
    double & x0,
    double & y0,
    double & r0,
    double & x1,
    double & y1,
    double & r1) const
```

Gets the gradient endpoint circles for a radial gradient, each specified as a center coordinate and a radius.

**Parameters**

<i>x0</i>	return value for the x coordinate of the center of the first (inner) circle
<i>y0</i>	return value for the y coordinate of the center of the first (inner) circle
<i>r0</i>	return value for the radius of the first (inner) circle
<i>x1</i>	return value for the x coordinate of the center of the second (outer) circle
<i>y1</i>	return value for the y coordinate of the center of the second (outer) circle
<i>r1</i>	return value for the radius of the second (outer) circle

**Since**

1.4

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

- `cairomm/pattern.h`

## 8.22 Cairo::RecordingSurface Class Reference

A recording surface is a surface that records all drawing operations at the highest level of the surface backend interface, (that is, the level of paint, mask, stroke, fill, and `show_text_glyphs`).

```
#include <cairomm/surface.h>
```

Inheritance diagram for Cairo::RecordingSurface:

**Public Member Functions**

- [RecordingSurface](#) (`cairo_surface_t *cobject`, `bool has_reference=false`)  
*Create a C++ wrapper for the C instance.*
- virtual [~RecordingSurface](#) ()
- [Rectangle ink\\_extents](#) () const  
*Measures the extents of the operations stored within the recording surface.*
- bool [get\\_extents](#) ([Rectangle](#) &extents) const  
*Get the extents of the recording surface, if the surface is bounded.*



## Public Member Functions inherited from Cairo::Surface

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & [operator=](#) (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* **data**, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const

- Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.
- void [write\\_to\\_png](#) (const **std::string** &filename)

Writes the contents of surface to a new file filename as a PNG image.
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)

Writes the [Surface](#) to the write function.
- [RefPtr< Device >](#) [get\\_device](#) ()

This function returns the device for a surface.
- [cobject](#) \* [cobj](#) ()

Provides acces to the underlying C cairo surface.
- const [cobject](#) \* [cobj](#) () const

Provides acces to the underlying C cairo surface.

### Static Public Member Functions

- static [RefPtr< RecordingSurface >](#) [create](#) ([Content](#) content=[Content::CONTENT\\_COLOR\\_ALPHA](#))

Creates a recording surface which can be used to record all drawing operations at the highest level (that is, the level of paint, mask, stroke, fill and [show\\_text\\_glyphs](#)).
- static [RefPtr< RecordingSurface >](#) [create](#) (const [Rectangle](#) &extents, [Content](#) content=[Content::CONTENT\\_COLOR\\_ALPHA](#))

Creates a recording surface which can be used to record all drawing operations at the highest level (that is, the level of paint, mask, stroke, fill and [show\\_text\\_glyphs](#)).

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) other, [Content](#) content, int width, int height)

Create a new surface that is as compatible as possible with an existing surface.
- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)

Create a new surface that is a rectangle within the target surface.

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {

[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,

[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,

[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,

[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,

[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,

[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,

[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,

[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,

[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,

[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,

[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,

[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,

[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,

[WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,

[QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,

[SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,

[QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,

```

RECORDING = CAIRO_SURFACE_TYPE_RECORDING ,
VG = CAIRO_SURFACE_TYPE_VG ,
GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)

*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*

- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)

*This is the type of function which is called when a backend needs to read data from an input stream.*

- typedef sigc::slot< void()> [SlotDestroy](#)

*For instance, void on\_destroy();.*

- typedef cairo\_surface\_t [cobject](#)

*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)

*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.22.1 Detailed Description

A recording surface is a surface that records all drawing operations at the highest level of the surface backend interface, (that is, the level of paint, mask, stroke, fill, and show\_text\_glyphs).

The recording surface can then be "replayed" against any target surface by using it as a source surface.

If you want to replay a surface so that the results in `target` will be identical to the results that would have been obtained if the original operations applied to the recording surface had instead been applied to the target surface, you can use code like this:

```

Cairo::RefPtr<Cairo::Context> context = Cairo::Context::create(target);
context->set_source(recording_surface, 0.0, 0.0);
context->paint();

```

A recording surface is logically unbounded, i.e. it has no implicit constraint on the size of the drawing surface. However, in practice this is rarely useful as you wish to replay against a particular target surface with known bounds. For this case, it is more efficient to specify the target extents to the recording surface upon creation.

The recording phase of the recording surface is careful to snapshot all necessary objects (paths, patterns, etc.), in order to achieve accurate replay.

Note that like all surfaces, a [RecordingSurface](#) is a reference-counted object that should be used via [Cairo::RefPtr](#).

## 8.22.2 Constructor & Destructor Documentation

### 8.22.2.1 RecordingSurface()

```
Cairo::RecordingSurface::RecordingSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.22.2.2 ~RecordingSurface()

```
virtual Cairo::RecordingSurface::~~RecordingSurface () [virtual]
```

## 8.22.3 Member Function Documentation

### 8.22.3.1 create() [1/2]

```
static RefPtr< RecordingSurface > Cairo::RecordingSurface::create (
    const Rectangle & extents,
    Content content = Content::CONTENT_COLOR_ALPHA) [static]
```

Creates a recording surface which can be used to record all drawing operations at the highest level (that is, the level of paint, mask, stroke, fill and show\_text\_glyphs).

The recording surface can then be "replayed" against any target surface by using it as a source to drawing operations. The recording surface will be bounded by the given rectangle.

The recording phase of the recording surface is careful to snapshot all necessary objects (paths, patterns, etc.), in order to achieve accurate replay.

#### Parameters

<i>extents</i>	the extents to record
<i>content</i>	the content of the recording surface; defaults to <a href="#">Cairo::Content::CONTENT_COLOR_ALPHA</a> .

#### Returns

a RefPtr to the newly created recording surface.

### 8.22.3.2 create() [2/2]

```
static RefPtr< RecordingSurface > Cairo::RecordingSurface::create (
    Content content = Content::CONTENT_COLOR_ALPHA) [static]
```

Creates a recording surface which can be used to record all drawing operations at the highest level (that is, the level of paint, mask, stroke, fill and show\_text\_glyphs).

The recording surface can then be "replayed" against any target surface by using it as a source to drawing operations. The recording surface will be unbounded.

The recording phase of the recording surface is careful to snapshot all necessary objects (paths, patterns, etc.), in order to achieve accurate replay.

## Parameters

<i>content</i>	the content of the recording surface; defaults to <a href="#">Cairo::Content::CONTENT_COLOR_ALPHA</a> .
----------------	---------------------------------------------------------------------------------------------------------

## Returns

a RefPtr to the newly created recording surface.

## 8.22.3.3 get\_extents()

```
bool Cairo::RecordingSurface::get_extents (
    Rectangle & extents) const
```

Get the extents of the recording surface, if the surface is bounded.

## Parameters

<i>extents</i>	the Rectangle in which to store the extents if the recording surface is bounded
----------------	---------------------------------------------------------------------------------

## Returns

true if the recording surface is bounded, false if the recording surface is unbounded (in which case *extents* will not be set).

## 8.22.3.4 ink\_extents()

```
Rectangle Cairo::RecordingSurface::ink_extents () const
```

Measures the extents of the operations stored within the recording surface.

This is useful to compute the required size of an image surface (or equivalent) into which to replay the full sequence of drawing operations.

## Returns

a Rectangle with *x* and *y* set to the coordinates of the top-left of the ink bounding box, and *width* and *height* set to the width and height of the ink bounding box.

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

- cairomm/surface.h

## 8.23 Cairo::Region Class Reference

A simple graphical data type representing an area of integer-aligned rectangles.

```
#include <cairomm/region.h>
```

## Public Types

- enum class [Overlap](#) {  
[IN](#) = CAIRO\_REGION\_OVERLAP\_IN ,  
[OUT](#) = CAIRO\_REGION\_OVERLAP\_OUT ,  
[REGION\\_OVERLAP\\_PART](#) = CAIRO\_REGION\_OVERLAP\_PART }
- typedef cairo\_region\_t [cobject](#)

## Public Member Functions

- [Region](#) (cairo\_region\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [RefPtr](#)< [Region](#) > [copy](#) () const  
*allocates a new region object copied from the original*
- virtual [~Region](#) ()
- [RectangleInt](#) [get\\_extents](#) () const  
*Gets the bounding rectangle of the region.*
- int [get\\_num\\_rectangles](#) () const  
*Gets the number of rectangles contained in the region.*
- [RectangleInt](#) [get\\_rectangle](#) (int nth\_rectangle) const  
*Gets the nth rectangle from the region.*
- bool [empty](#) () const  
*Checks whether the region is empty.*
- [Overlap](#) [contains\\_rectangle](#) (const [RectangleInt](#) &rectangle) const  
*Checks whether rectangle is inside, outside, or partially contained in the region.*
- bool [contains\\_point](#) (int x, int y) const  
*Checks whether (x,y) is contained in the region.*
- void [translate](#) (int dx, int dy)  
*Translates the region by (dx,dy)*
- void [subtract](#) (const [RefPtr](#)< [Region](#) > &other)  
*Subtracts other from this region.*
- void [subtract](#) (const [RectangleInt](#) &rectangle)  
*Subtracts rectangle from this region.*
- void [intersect](#) (const [RefPtr](#)< [Region](#) > &other)  
*Sets the region to the intersection of this region with other.*
- void [intersect](#) (const [RectangleInt](#) &rectangle)  
*Sets the region to the intersection of this region with rectangle.*
- void [do\\_union](#) (const [RefPtr](#)< [Region](#) > &other)  
*Sets this region to the union of the region with other.*
- void [do\\_union](#) (const [RectangleInt](#) &rectangle)  
*Sets this region to the union of the region with rectangle.*
- void [do\\_xor](#) (const [RefPtr](#)< [Region](#) > &other)  
*Sets this region to the exclusive difference of the region with other.*
- void [do\\_xor](#) (const [RectangleInt](#) &rectangle)  
*Sets this region to the exclusive difference of the region with rectangle.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

### Static Public Member Functions

- static [RefPtr< Region > create](#) ()  
*Creates an empty [Region](#) object.*
- static [RefPtr< Region > create](#) (const [RectangleInt](#) &rectangle)  
*Creates a [Region](#) object containing rectangle.*
- static [RefPtr< Region > create](#) (const **std::vector**< [RectangleInt](#) > &rects)  
*Creates a [Region](#) object containing the union of all given rects.*
- static [RefPtr< Region > create](#) (const [RectangleInt](#) \*rects, int **count**)  
*Creates a [Region](#) object containing the union of all given rects.*

### Protected Attributes

- [cobject](#) \* [m\\_cobject](#)

## 8.23.1 Detailed Description

A simple graphical data type representing an area of integer-aligned rectangles.

They are often used on raster surfaces to track areas of interest, such as change or clip areas

It allows set-theoretical operations like union and intersect to be performed on them.

Since

: 1.10

## 8.23.2 Member Typedef Documentation

### 8.23.2.1 cobject

```
typedef cairo_region_t Cairo::Region::cobject
```

## 8.23.3 Member Enumeration Documentation

### 8.23.3.1 Overlap

```
enum class Cairo::Region::Overlap [strong]
```

Enumerator

IN	Completely inside region.
OUT	Completely outside region.
REGION_OVERLAP_PART	Partly inside region.

## 8.23.4 Constructor & Destructor Documentation

### 8.23.4.1 Region()

```
Cairo::Region::Region (  
    cairo_region_t * cobject,  
    bool has\_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a [RefPtr](#).

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

**8.23.4.2 ~Region()**

```
virtual Cairo::Region::~~Region () [virtual]
```

**8.23.5 Member Function Documentation****8.23.5.1 cobj() [1/2]**

```
cobject * Cairo::Region::cobj () [inline]
```

**8.23.5.2 cobj() [2/2]**

```
const cobject * Cairo::Region::cobj () const [inline]
```

**8.23.5.3 contains\_point()**

```
bool Cairo::Region::contains_point (
    int x,
    int y) const
```

Checks whether (x,y) is contained in the region.

**8.23.5.4 contains\_rectangle()**

```
Overlap Cairo::Region::contains_rectangle (
    const RectangleInt & rectangle) const
```

Checks whether *rectangle* is inside, outside, or partially contained in the region.

**8.23.5.5 copy()**

```
RefPtr< Region > Cairo::Region::copy () const
```

allocates a new region object copied from the original

**8.23.5.6 create() [1/4]**

```
static RefPtr< Region > Cairo::Region::create () [static]
```

Creates an empty [Region](#) object.



**8.23.5.7 create() [2/4]**

```
static RefPtr< Region > Cairo::Region::create (  
    const RectangleInt & rectangle) [static]
```

Creates a [Region](#) object containing *rectangle*.

**8.23.5.8 create() [3/4]**

```
static RefPtr< Region > Cairo::Region::create (  
    const RectangleInt * rects,  
    int count) [static]
```

Creates a [Region](#) object containing the union of all given *rects*.

**8.23.5.9 create() [4/4]**

```
static RefPtr< Region > Cairo::Region::create (  
    const std::vector< RectangleInt > & rects) [static]
```

Creates a [Region](#) object containing the union of all given *rects*.

**8.23.5.10 do\_union() [1/2]**

```
void Cairo::Region::do_union (  
    const RectangleInt & rectangle)
```

Sets this region to the union of the region with *rectangle*.

**8.23.5.11 do\_union() [2/2]**

```
void Cairo::Region::do_union (  
    const RefPtr< Region > & other)
```

Sets this region to the union of the region with *other*.

**8.23.5.12 do\_xor() [1/2]**

```
void Cairo::Region::do_xor (  
    const RectangleInt & rectangle)
```

Sets this region to the exclusive difference of the region with *rectangle*.

That is, the region will contain all areas that are in the original region or in *rectangle*, but not in both

#### 8.23.5.13 do\_xor() [2/2]

```
void Cairo::Region::do_xor (
    const RefPtr< Region > & other)
```

Sets this region to the exclusive difference of the region with *other*.

That is, the region will contain all areas that are in the original region or in *other*, but not in both

#### 8.23.5.14 empty()

```
bool Cairo::Region::empty () const
```

Checks whether the region is empty.

#### 8.23.5.15 get\_extents()

```
RectangleInt Cairo::Region::get_extents () const
```

Gets the bounding rectangle of the region.

#### 8.23.5.16 get\_num\_rectangles()

```
int Cairo::Region::get_num_rectangles () const
```

Gets the number of rectangles contained in the region.

#### 8.23.5.17 get\_rectangle()

```
RectangleInt Cairo::Region::get_rectangle (
    int nth_rectangle) const
```

Gets the nth rectangle from the region.

#### 8.23.5.18 intersect() [1/2]

```
void Cairo::Region::intersect (
    const RectangleInt & rectangle)
```

Sets the region to the intersection of this region with *rectangle*.

#### 8.23.5.19 intersect() [2/2]

```
void Cairo::Region::intersect (
    const RefPtr< Region > & other)
```

Sets the region to the intersection of this region with *other*.

#### 8.23.5.20 reference()

```
void Cairo::Region::reference () const
```

#### 8.23.5.21 subtract() [1/2]

```
void Cairo::Region::subtract (  
    const RectangleInt & rectangle)
```

Subtracts *rectangle* from this region.

#### 8.23.5.22 subtract() [2/2]

```
void Cairo::Region::subtract (  
    const RefPtr< Region > & other)
```

Subtracts *other* from this region.

#### 8.23.5.23 translate()

```
void Cairo::Region::translate (  
    int dx,  
    int dy)
```

Translates the region by (dx,dy)

#### 8.23.5.24 unreference()

```
void Cairo::Region::unreference () const
```

### 8.23.6 Member Data Documentation

#### 8.23.6.1 m\_cobject

```
cobject* Cairo::Region::m_cobject [protected]
```

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

- cairomm/region.h

## 8.24 Cairo::SaveGuard Class Reference

RAII-style context save/restore class.

```
#include <cairomm/context.h>
```

## Public Member Functions

- [SaveGuard](#) (const [RefPtr](#)< [Context](#) > &context)  
*Constructor, the context is saved.*
- [~SaveGuard](#) ()  
*Destructor, the context is restored.*

### 8.24.1 Detailed Description

RAII-style context save/restore class.

[Cairo::Context::save\(\)](#) is called automatically when the object is created, and [Cairo::Context::restore\(\)](#) is called when the object is destroyed. This allows you to write code such as:

```
// context initial state
{
    Cairo::SaveGuard saver(context);
    ... // manipulate context
}
// context is restored to initial state
```

Since [caiomm 1.18](#)

### 8.24.2 Constructor & Destructor Documentation

#### 8.24.2.1 SaveGuard()

```
Cairo::SaveGuard::SaveGuard (
    const RefPtr< Context > & context) [explicit]
```

Constructor, the context is saved.

#### 8.24.2.2 ~SaveGuard()

```
Cairo::SaveGuard::~~SaveGuard ()
```

Destructor, the context is restored.

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

- [caiomm/context.h](#)

## 8.25 Cairo::ScaledFont Class Reference

A [ScaledFont](#) is a font scaled to a particular size and device resolution.

```
#include <cairomm/scaledfont.h>
```

Inheritance diagram for Cairo::ScaledFont:



### Public Types

- typedef cairo\_scaled\_font\_t [cobject](#)  
The underlying C cairo object type.

### Public Member Functions

- [cobject](#) \* [cobj](#) ()  
Provides acces to the underlying C cairo object.
- const [cobject](#) \* [cobj](#) () const  
Provides acces to the underlying C cairo object.
- [ScaledFont](#) ([cobject](#) \*[cobj](#), bool has\_reference=false)  
Create a C++ wrapper object from the C instance.
- [ScaledFont](#) (const [ScaledFont](#) &)=delete
- [ScaledFont](#) & [operator=](#) (const [ScaledFont](#) &)=delete
- virtual ~[ScaledFont](#) ()
- void [get\\_extents](#) ([FontExtents](#) &extents) const  
Gets the metrics for a [ScaledFont](#).
- void [get\\_text\\_extents](#) (const **std::string** &utf8, [TextExtents](#) &extents) const  
Gets the extents for a string of text.
- void [get\\_glyph\\_extents](#) (const **std::vector**< [Glyph](#) > &glyphs, [TextExtents](#) &extents) const  
Gets the extents for an array of glyphs.
- [RefPtr](#)< [FontFace](#) > [get\\_font\\_face](#) () const  
The [FontFace](#) with which this [ScaledFont](#) was created.
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
Gets the [FontOptions](#) with which the [ScaledFont](#) was created.
- void [get\\_font\\_matrix](#) ([Matrix](#) &font\_matrix) const  
Gets the font matrix with which the [ScaledFont](#) was created.

- void `get_ctm` (`Matrix` &ctm) const  
*Gets the CTM with which the `ScaledFont` was created.*
- `FontType` `get_type` () const  
*Gets the type of scaled Font.*
- void `text_to_glyphs` (double x, double y, const `std::string` &utf8, `std::vector`< `Glyph` > &glyphs, `std::vector`< `TextCluster` > &clusters, `TextClusterFlags` &cluster\_flags)
- void `get_scale_matrix` (`Matrix` &scale\_matrix) const  
*Stores the scale matrix of this scaled font into matrix.*

### Static Public Member Functions

- static `RefPtr`< `ScaledFont` > `create` (const `RefPtr`< `FontFace` > &font\_face, const `Matrix` &font\_matrix, const `Matrix` &ctm, const `FontOptions` &options=`FontOptions`())  
*Creates a `ScaledFont` object from a font face and matrices that describe the size of the font and the environment in which it will be used.*

### Protected Member Functions

- `ScaledFont` (const `RefPtr`< `FontFace` > &font\_face, const `Matrix` &font\_matrix, const `Matrix` &ctm, const `FontOptions` &options=`FontOptions`())

### Protected Attributes

- `cobject` \* `m_cobject`  
*The underlying C cairo object that is wrapped by this `ScaledFont`.*

## 8.25.1 Detailed Description

A `ScaledFont` is a font scaled to a particular size and device resolution.

It is most useful for low-level font usage where a library or application wants to cache a reference to a scaled font to speed up the computation of metrics.

## 8.25.2 Member Typedef Documentation

### 8.25.2.1 cobject

```
typedef cairo_scaled_font_t Cairo::ScaledFont::cobject
```

The underlying C cairo object type.

## 8.25.3 Constructor & Destructor Documentation

### 8.25.3.1 ScaledFont() [1/3]

```
Cairo::ScaledFont::ScaledFont (
    cobject * cobj,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper object from the C instance.

This C++ object should then be given to a `RefPtr`.

### 8.25.3.2 ScaledFont() [2/3]

```
Cairo::ScaledFont::ScaledFont (
    const ScaledFont & ) [delete]
```

### 8.25.3.3 ~ScaledFont()

```
virtual Cairo::ScaledFont::~~ScaledFont () [virtual]
```

### 8.25.3.4 ScaledFont() [3/3]

```
Cairo::ScaledFont::ScaledFont (
    const RefPtr< FontFace > & font_face,
    const Matrix & font_matrix,
    const Matrix & ctm,
    const FontOptions & options = FontOptions()) [protected]
```

## 8.25.4 Member Function Documentation

### 8.25.4.1 cobj() [1/2]

```
cobject * Cairo::ScaledFont::cobj () [inline]
```

Provides acces to the underlying C cairo object.

### 8.25.4.2 cobj() [2/2]

```
const cobject * Cairo::ScaledFont::cobj () const [inline]
```

Provides acces to the underlying C cairo object.

### 8.25.4.3 create()

```
static RefPtr< ScaledFont > Cairo::ScaledFont::create (
    const RefPtr< FontFace > & font_face,
    const Matrix & font_matrix,
    const Matrix & ctm,
    const FontOptions & options = FontOptions()) [static]
```

Creates a [ScaledFont](#) object from a font face and matrices that describe the size of the font and the environment in which it will be used.

#### Parameters

<i>font_face</i>	A font face.
<i>font_matrix</i>	font space to user space transformation matrix for the font. In the simplest case of a N point font, this matrix is just a scale by N, but it can also be used to shear the font or stretch it unequally along the two axes. See <a href="#">Context::set_font_matrix()</a> .
<i>ctm</i>	user to device transformation matrix with which the font will be used.
<i>options</i>	options to use when getting metrics for the font and rendering with it.

#### 8.25.4.4 get\_ctm()

```
void Cairo::ScaledFont::get_ctm (
    Matrix & ctm) const
```

Gets the CTM with which the [ScaledFont](#) was created.

Since

1.2

#### 8.25.4.5 get\_extents()

```
void Cairo::ScaledFont::get_extents (
    FontExtents & extents) const
```

Gets the metrics for a [ScaledFont](#).

Since

1.8

#### 8.25.4.6 get\_font\_face()

```
RefPtr< FontFace > Cairo::ScaledFont::get_font_face () const
```

The [FontFace](#) with which this [ScaledFont](#) was created.

Since

1.2

#### 8.25.4.7 get\_font\_matrix()

```
void Cairo::ScaledFont::get_font_matrix (
    Matrix & font_matrix) const
```

Gets the font matrix with which the [ScaledFont](#) was created.

Since

1.2



#### 8.25.4.8 get\_font\_options()

```
void Cairo::ScaledFont::get_font_options (
    FontOptions & options) const
```

Gets the [FontOptions](#) with which the [ScaledFont](#) was created.

Since

1.2

#### 8.25.4.9 get\_glyph\_extents()

```
void Cairo::ScaledFont::get_glyph_extents (
    const std::vector< Glyph > & glyphs,
    TextExtents & extents) const
```

Gets the extents for an array of glyphs.

The extents describe a user-space rectangle that encloses the "inked" portion of the glyphs, (as they would be drawn by [Context::show\\_glyphs\(\)](#) if the cairo graphics state were set to the same font\_face, font\_matrix, ctm, and font\_options as the [ScaledFont](#) object). Additionally, the x\_advance and y\_advance values indicate the amount by which the current point would be advanced by [Context::show\\_glyphs\(\)](#).

Note that whitespace glyphs do not contribute to the size of the rectangle (extents.width and extents.height).

Parameters

<i>glyphs</i>	A vector of glyphs to calculate the extents of.
<i>extents</i>	Returns the extents for the array of glyphs.

Since

1.8

#### 8.25.4.10 get\_scale\_matrix()

```
void Cairo::ScaledFont::get_scale_matrix (
    Matrix & scale_matrix) const
```

Stores the scale matrix of this scaled font into matrix.

The scale matrix is product of the font matrix and the ctm associated with the scaled font, and hence is the matrix mapping from font space to device space.

Parameters

<i>scale_matrix</i>	return value for the matrix.
---------------------	------------------------------

Since

1.8

#### 8.25.4.11 get\_text\_extents()

```
void Cairo::ScaledFont::get_text_extents (
    const    std::string & utf8,
    TextExtents & extents) const
```

Gets the extents for a string of text.

The extents describe a user-space rectangle that encloses the "inked" portion of the text drawn at the origin (0,0) (as it would be drawn by [Context::show\\_text\(\)](#) if the cairo graphics state were set to the same font\_face, font\_matrix, ctm, and font\_options as the [ScaledFont](#) object). Additionally, the x\_advance and y\_advance values indicate the amount by which the current point would be advanced by [Context::show\\_text\(\)](#).

Note that whitespace characters do not directly contribute to the size of the rectangle (extents.width and extents.height). They do contribute indirectly by changing the position of non-whitespace characters. In particular, trailing whitespace characters are likely to not affect the size of the rectangle, though they will affect the x\_advance and y\_advance values.

##### Parameters

<i>utf8</i>	a string of text, encoded in UTF-8.
<i>extents</i>	Returns the extents of the given string.

##### Since

1.8

#### 8.25.4.12 get\_type()

```
FontType Cairo::ScaledFont::get_type () const
```

Gets the type of scaled Font.

##### Since

1.2

#### 8.25.4.13 operator=()

```
ScaledFont & Cairo::ScaledFont::operator= (
    const ScaledFont & ) [delete]
```

#### 8.25.4.14 text\_to\_glyphs()

```
void Cairo::ScaledFont::text_to_glyphs (
    double x,
    double y,
    const    std::string & utf8,
    std::vector< Glyph > & glyphs,
    std::vector< TextCluster > & clusters,
    TextClusterFlags & cluster_flags)
```

## Parameters

<i>x</i>	X position to place first glyph.
<i>y</i>	Y position to place first glyph.
<i>utf8</i>	a string of text encoded in UTF-8.
<i>glyphs</i>	pointer to array of glyphs to fill.
<i>clusters</i>	pointer to array of cluster mapping information to fill. @cluster_flags cluster mapping flags

Converts UTF-8 text to an array of glyphs, with cluster mapping, that can be used to render later.

For details of how (*clusters* and *cluster\_flags* map input UTF-8 text to the output glyphs see [Context::show\\_text\\_glyphs\(\)](#).

The output values can be readily passed to [Context::show\\_text\\_glyphs\(\)](#) [Context::show\\_glyphs\(\)](#), or related functions, assuming that the exact same scaled font is used for the operation.

Since

1.8

## 8.25.5 Member Data Documentation

### 8.25.5.1 m\_cobject

`cobject*` Cairo::ScaledFont::m\_cobject [protected]

The underlying C cairo object that is wrapped by this [ScaledFont](#).

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

- cairomm/scaledfont.h

## 8.26 Cairo::SolidPattern Class Reference

```
#include <cairomm/pattern.h>
```

Inheritance diagram for Cairo::SolidPattern:



### Public Member Functions

- [SolidPattern](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- void [get\\_rgba](#) (double &red, double &green, double &blue, double &alpha) const  
*Gets the solid color for a solid color pattern.*
- [~SolidPattern](#) () override

### Public Member Functions inherited from [Cairo::Pattern](#)

- [Pattern](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Pattern](#) (const [Pattern](#) &)=delete
- [Pattern](#) & operator= (const [Pattern](#) &)=delete
- virtual [~Pattern](#) ()
- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the pattern's transformation matrix to @matrix.*
- void [get\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the pattern's transformation matrix.*
- [Matrix](#) [get\\_matrix](#) () const  
*Returns the pattern's transformation matrix.*
- [Type](#) [get\\_type](#) () const  
*Returns the type of the pattern.*
- void [set\\_extend](#) ([Extend](#) extend)  
*Sets the mode to be used for drawing outside the area of a pattern.*
- [Extend](#) [get\\_extend](#) () const  
*Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

### Static Public Member Functions

- static [RefPtr](#)< [SolidPattern](#) > [create\\_rgb](#) (double red, double green, double blue)  
*Creates a new Cairo::Pattern corresponding to an opaque color.*
- static [RefPtr](#)< [SolidPattern](#) > [create\\_rgba](#) (double red, double green, double blue, double alpha)  
*Creates a new Cairo::Pattern corresponding to a translucent color.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Pattern](#)

- enum class [Type](#) {  
  [SOLID](#) = CAIRO\_PATTERN\_TYPE\_SOLID ,  
  [SURFACE](#) = CAIRO\_PATTERN\_TYPE\_SURFACE ,  
  [LINEAR](#) = CAIRO\_PATTERN\_TYPE\_LINEAR ,  
  [RADIAL](#) = CAIRO\_PATTERN\_TYPE\_RADIAL }  
*Type is used to describe the type of a given pattern.*
- enum class [Extend](#) {  
  [NONE](#) = CAIRO\_EXTEND\_NONE ,  
  [REPEAT](#) = CAIRO\_EXTEND\_REPEAT ,  
  [REFLECT](#) = CAIRO\_EXTEND\_REFLECT ,  
  [PAD](#) = CAIRO\_EXTEND\_PAD }  
*Cairo::Extend is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).*
- typedef cairo\_pattern\_t [cobject](#)

## Protected Member Functions inherited from Cairo::Pattern

- [Pattern\(\)](#)

## Protected Attributes inherited from Cairo::Pattern

- [cobject](#) \* [m\\_cobject](#)

## 8.26.1 Constructor & Destructor Documentation

### 8.26.1.1 SolidPattern()

```
Cairo::SolidPattern::SolidPattern (
    cairo_pattern_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.26.1.2 ~SolidPattern()

```
Cairo::SolidPattern::~~SolidPattern () [override]
```

## 8.26.2 Member Function Documentation

### 8.26.2.1 create\_rgb()

```
static RefPtr< SolidPattern > Cairo::SolidPattern::create_rgb (
    double red,
    double green,
    double blue) [static]
```

Creates a new [Cairo::Pattern](#) corresponding to an opaque color.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

#### Parameters

<i>red</i>	red component of the color
<i>green</i>	green component of the color
<i>blue</i>	blue component of the color

### 8.26.2.2 create\_rgba()

```
static RefPtr< SolidPattern > Cairo::SolidPattern::create_rgba (
    double red,
    double green,
    double blue,
    double alpha) [static]
```

Creates a new [Cairo::Pattern](#) corresponding to a translucent color.

The color components are floating point numbers in the range 0 to 1. If the values passed in are outside that range, they will be clamped.

#### Parameters

<i>red</i>	red component of the color
<i>green</i>	green component of the color
<i>blue</i>	blue component of the color
<i>alpha</i>	alpha component of the color

### 8.26.2.3 get\_rgba()

```
void Cairo::SolidPattern::get_rgba (
    double & red,
    double & green,
    double & blue,
    double & alpha) const
```

Gets the solid color for a solid color pattern.

#### Parameters

<i>red</i>	return value for red component of color
<i>green</i>	return value for green component of color
<i>blue</i>	return value for blue component of color
<i>alpha</i>	return value for alpha component of color

#### Since

1.4

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

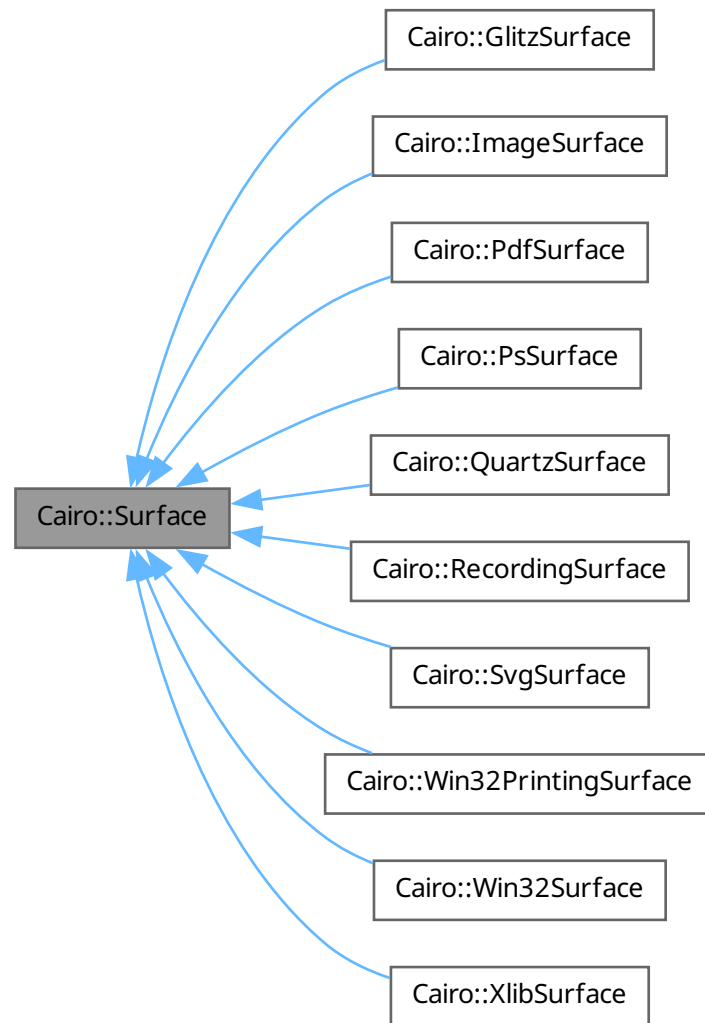
- [caiomm/pattern.h](#)

## 8.27 Cairo::Surface Class Reference

A cairo surface represents an image, either as the destination of a drawing operation or as source when drawing onto another surface.

```
#include <caiomm/surface.h>
```

Inheritance diagram for Cairo::Surface:



### Public Types

- enum class [Type](#) {  
    [IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
    [PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
    [PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
    [XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,

```

XCB = CAIRO_SURFACE_TYPE_XCB ,
GLITZ = CAIRO_SURFACE_TYPE_GLITZ ,
QUARTZ = CAIRO_SURFACE_TYPE_QUARTZ ,
WIN32 = CAIRO_SURFACE_TYPE_WIN32 ,
WIN32_SURFACE = CAIRO_SURFACE_TYPE_WIN32 ,
BEOS = CAIRO_SURFACE_TYPE_BEOS ,
DIRECTFB = CAIRO_SURFACE_TYPE_DIRECTFB ,
SVG = CAIRO_SURFACE_TYPE_SVG ,
OS2 = CAIRO_SURFACE_TYPE_OS2 ,
WIN32_PRINTING = CAIRO_SURFACE_TYPE_WIN32_PRINTING ,
QUARTZ_IMAGE = CAIRO_SURFACE_TYPE_QUARTZ_IMAGE ,
SCRIPT = CAIRO_SURFACE_TYPE_SCRIPT ,
QT = CAIRO_SURFACE_TYPE_QT ,
RECORDING = CAIRO_SURFACE_TYPE_RECORDING ,
VG = CAIRO_SURFACE_TYPE_VG ,
GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }  
*Format is used to identify the memory format of image data.*
- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)  
*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*
- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)  
*This is the type of function which is called when a backend needs to read data from an input stream.*
- typedef sigc::slot< void()> [SlotDestroy](#)  
*For instance, void on\_destroy();.*
- typedef cairo\_surface\_t [cobject](#)  
*The underlying C cairo surface type.*

## Public Member Functions

- [Surface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & operator= (const [Surface](#) &)=delete
- virtual ~[Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* **data**, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*



- void [finish](#) ()
 

*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()
 

*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()
 

*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)
 

*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)
 

*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const
 

*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)
 

*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)
 

*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const
 

*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const
 

*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)
 

*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const
 

*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const
 

*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()
 

*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()
 

*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const
 

*Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)
 

*Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)
 

*Writes the [Surface](#) to the write function.*
- [RefPtr](#)< [Device](#) > [get\\_device](#) ()
 

*This function returns the device for a surface.*
- [cobject](#) \* [cobj](#) ()
 

*Provides acces to the underlying C cairo surface.*
- const [cobject](#) \* [cobj](#) () const
 

*Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr< Surface > create](#) (const [RefPtr< Surface >](#) other, [Content](#) content, int width, int height)  
*Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr< Surface > create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)  
*Create a new surface that is a rectangle within the target surface.*

### Protected Attributes

- [cobject](#) \* [m\\_cobject](#)  
*The underlying C cairo surface type that is wrapped by this [Surface](#).*

## 8.27.1 Detailed Description

A cairo surface represents an image, either as the destination of a drawing operation or as source when drawing onto another surface.

There are different subtypes of cairo surface for different drawing backends. This class is a base class for all subtypes and should not be used directly

Most surface types allow accessing the surface without using [Cairo](#) functions. If you do this, keep in mind that it is mandatory that you call [Cairo::Surface::flush\(\)](#) before reading from or writing to the surface and that you must use [Cairo::Surface::mark\\_dirty\(\)](#) after modifying it.

Surfaces are reference-counted objects that should be used via [Cairo::RefPtr](#).

## 8.27.2 Member Typedef Documentation

### 8.27.2.1 cobject

```
typedef cairo_surface_t Cairo::Surface::cobject
```

The underlying C cairo surface type.

### 8.27.2.2 SlotDestroy

```
typedef sigc::slot<void()> Cairo::Surface::SlotDestroy
```

For instance, void on\_destroy();.

### 8.27.2.3 SlotReadFunc

```
typedef sigc::slot<ErrorStatus(unsigned char* , unsigned int )> Cairo::Surface::SlotReadFunc
```

This is the type of function which is called when a backend needs to read data from an input stream.

It is passed the buffer to read the data into and the length of the data in bytes. The read function should return CAIRO\_STATUS\_SUCCESS if all the data was successfully read, CAIRO\_STATUS\_READ\_ERROR otherwise.

## Parameters

<i>data</i>	the buffer into which to read the data
<i>length</i>	the amount of data to read

## Returns

the status code of the read operation

## 8.27.2.4 SlotWriteFunc

```
typedef sigc::slot<ErrorStatus(const unsigned char* , unsigned int )> Cairo::Surface::SlotWriteFunc
```

For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`

This is the type of function which is called when a backend needs to write data to an output stream. It is passed the data to write and the length of the data in bytes. The write function should return `CAIRO_STATUS_SUCCESS` if all the data was successfully written, `CAIRO_STATUS_WRITE_ERROR` otherwise.

## Parameters

<i>data</i>	the buffer containing the data to write
<i>length</i>	the amount of data to write

## Returns

the status code of the write operation

## 8.27.3 Member Enumeration Documentation

## 8.27.3.1 Format

```
enum class Cairo::Surface::Format [strong]
```

Format is used to identify the memory format of image data.

New entries may be added in future versions.

## Enumerator

ARGB32	Each pixel is a 32-bit quantity, with alpha in the upper 8 bits, then red, then green, then blue. The 32-bit quantities are stored native-endian. Pre-multiplied alpha is used. (That is, 50% transparent red is 0x80800000,
RGB24	Each pixel is a 32-bit quantity, with the upper 8 bits unused. Red, Green, and Blue are stored in the remaining 24 bits in that order.
A8	Each pixel is a 8-bit quantity holding an alpha value.
A1	Each pixel is a 1-bit quantity holding an alpha value. Pixels are packed together into 32-bit quantities. The ordering of the bits matches the endianness of the platform. On a big-endian machine, the first pixel is in the uppermost bit, on a little endian machine the first pixel is in the least-significant bit.
RGB16_565	Each pixel is a 16-bit quantity with red in the upper 5 bits, then green in the middle 6 bits, and blue in the lower 5 bits.

### 8.27.3.2 Type

```
enum class Cairo::Surface::Type [strong]
```

Cairo::Surface::Type is used to describe the type of a given surface.

The surface types are also known as "backends" or "surface backends" within cairo.

The surface type can be queried with [Surface::get\\_type\(\)](#)

The various [Cairo::Surface](#) functions can be used with surfaces of any type, but some backends also provide type-specific functions that must only be called with a surface of the appropriate type.

New entries may be added in future versions.

Since

1.2

#### Enumerator

IMAGE	The surface is of type image.
PDF	The surface is of type pdf.
PS	The surface is of type ps.
XLIB	The surface is of type xlib.
XCB	The surface is of type xcb.
GLITZ	The surface is of type glitz.
QUARTZ	The surface is of type quartz.
WIN32	The surface is of type win32.  <b>Deprecated</b> Use WIN32_SURFACE instead.
WIN32_SURFACE	The surface is of type win32.  Since 1.16.1
BEOS	The surface is of type beos.
DIRECTFB	The surface is of type directfb.
SVG	The surface is of type svg.
OS2	The surface is of type os2.
WIN32_PRINTING	The surface is a win32 printing surface.
QUARTZ_IMAGE	The surface is of type quartz_image.
SCRIPT	The surface is of type script.  Since 1.10
QT	The surface is of type Qt.  Since 1.10

## Enumerator

RECORDING	The surface is of type recording.  Since 1.10
VG	The surface is a OpenVg surface.  Since 1.10
GL	The surface is of type OpenGL.  Since 1.10
DRM	The surface is of type Direct Render Manager.  Since 1.10
TEE	The surface is of type script 'tee' (a multiplexing surface)  Since 1.10
XML	The surface is of type XML (for debugging)  Since 1.10
SKIA	The surface is of type Skia.  Since 1.10
SUBSURFACE	The surface is of type The surface is a subsurface created with <a href="#">Surface::create()</a>  Since 1.10

## 8.27.4 Constructor & Destructor Documentation

### 8.27.4.1 Surface() [1/2]

```
Cairo::Surface::Surface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a RefPtr.

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

**8.27.4.2 Surface()** [2/2]

```
Cairo::Surface::Surface (
    const Surface & ) [delete]
```

**8.27.4.3 ~Surface()**

```
virtual Cairo::Surface::~~Surface () [virtual]
```

**8.27.5 Member Function Documentation****8.27.5.1 cobj()** [1/2]

```
cobject * Cairo::Surface::cobj () [inline]
```

Provides acces to the underlying C cairo surface.

**8.27.5.2 cobj()** [2/2]

```
const cobject * Cairo::Surface::cobj () const [inline]
```

Provides acces to the underlying C cairo surface.

**8.27.5.3 copy\_page()**

```
void Cairo::Surface::copy_page ()
```

Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.

Use [show\\_page\(\)](#) if you want to get an empty page after the emission.

Since

1.6

#### 8.27.5.4 create() [1/2]

```
static RefPtr< Surface > Cairo::Surface::create (
    const RefPtr< Surface > & target,
    double x,
    double y,
    double width,
    double height) [static]
```

Create a new surface that is a rectangle within the target surface.

All operations drawn to this surface are then clipped and translated onto the target surface. Nothing drawn via this sub-surface outside of its bounds is drawn onto the target surface, making this a useful method for passing constrained child surfaces to library routines that draw directly onto the parent surface, i.e. with no further backend allocations, double buffering or copies.

@Note The semantics of subsurfaces have not been finalized yet unless the rectangle is in full device units, is contained within the extents of the target surface, and the target or subsurface's device transforms are not changed.

## Parameters

<i>target</i>	an existing surface for which the sub-surface will point to
<i>x</i>	the x-origin of the sub-surface from the top-left of the target surface (in device-space units)
<i>y</i>	the y-origin of the sub-surface from the top-left of the target surface (in device-space units)
<i>width</i>	width of the sub-surface (in device-space units)
<i>height</i>	height of the sub-surface (in device-space units)

## Since

1.10

**8.27.5.5 create() [2/2]**

```
static RefPtr< Surface > Cairo::Surface::create (
    const RefPtr< Surface > other,
    Content content,
    int width,
    int height) [static]
```

Create a new surface that is as compatible as possible with an existing surface.

The new surface will use the same backend as other unless that is not possible for some reason.

## Parameters

<i>other</i>	an existing surface used to select the backend of the new surface
<i>content</i>	the content for the new surface
<i>width</i>	width of the new surface, (in device-space units)
<i>height</i>	height of the new surface (in device-space units)

## Returns

a RefPtr to the newly allocated surface.

**8.27.5.6 finish()**

```
void Cairo::Surface::finish ()
```

This function finishes the surface and drops all references to external resources.

For example, for the Xlib backend it means that cairo will no longer access the drawable, which can be freed. After calling [finish\(\)](#) the only valid operations on a surface are getting and setting user data and referencing and destroying it. Further drawing to the surface will not affect the surface but will instead trigger a CAIRO\_STATUS\_SURFACE↵\_FINISHED error.

When the [Surface](#) is destroyed, cairo will call [finish\(\)](#) if it hasn't been called already, before freeing the resources associated with the [Surface](#).



### 8.27.5.7 flush()

```
void Cairo::Surface::flush ()
```

Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.

This function must be called before switching from drawing on the surface with cairo to drawing on it directly with native APIs. If the surface doesn't support direct access, then this function does nothing.

### 8.27.5.8 get\_content()

```
Content Cairo::Surface::get_content () const
```

This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.

Since

1.8

### 8.27.5.9 get\_device()

```
RefPtr< Device > Cairo::Surface::get_device ()
```

This function returns the device for a surface.

Returns

The device for this surface, or an empty RefPtr if the surface has no associated device

### 8.27.5.10 get\_device\_offset()

```
void Cairo::Surface::get_device_offset (
    double & x_offset,
    double & y_offset) const
```

Returns a previous device offset set by [set\\_device\\_offset\(\)](#).

### 8.27.5.11 get\_device\_scale() [1/2]

```
double Cairo::Surface::get_device_scale () const
```

Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).

Since [cairomm 1.18](#)

**8.27.5.12 get\_device\_scale()** [2/2]

```
void Cairo::Surface::get_device_scale (
    double & x_scale,
    double & y_scale) const
```

Returns a previous device scale set by [set\\_device\\_scale\(\)](#).

Since [cairomm 1.18](#)

**8.27.5.13 get\_fallback\_resolution()**

```
void Cairo::Surface::get_fallback_resolution (
    double & x_pixels_per_inch,
    double & y_pixels_per_inch) const
```

This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.

**Parameters**

<i>x_pixels_per_inch</i>	horizontal pixels per inch
<i>y_pixels_per_inch</i>	vertical pixels per inch

Since

1.8

**8.27.5.14 get\_font\_options()**

```
void Cairo::Surface::get_font_options (
    FontOptions & options) const
```

Retrieves the default font rendering options for the surface.

This allows display surfaces to report the correct subpixel order for rendering on them, print surfaces to disable hinting of metrics and so forth. The result can then be used with [Cairo::ScaledFont::create\(\)](#).

**Parameters**

<i>options</i>	a <a href="#">FontOptions</a> object into which to store the retrieved options. All existing values are overwritten
----------------	---------------------------------------------------------------------------------------------------------------------

**8.27.5.15 get\_mime\_data()**

```
const unsigned char * Cairo::Surface::get_mime_data (
    const std::string & mime_type,
    unsigned long & length)
```

Return mime data previously attached to surface using the specified mime type.

If no data has been attached with the given mime type then this returns 0.

## Parameters

<i>mime_type</i>	The MIME type of the image data.
<i>length</i>	This will be set to the length of the image data.

## Returns

The image data attached to the surface.

## Since

1.10

**8.27.5.16 get\_type()**

```
Type Cairo::Surface::get_type () const
```

**8.27.5.17 has\_show\_text\_glyphs()**

```
bool Cairo::Surface::has_show_text_glyphs () const
```

Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.

That is, whether it actually uses the provided text and cluster data to a [Context::show\\_text\\_glyphs\(\)](#) call.

Note: Even if this function returns FALSE, a [Context::show\\_text\\_glyphs\(\)](#) operation targeted at this surface will still succeed. It just will act like a [Context::show\\_glyphs\(\)](#) operation. Users can use this function to avoid computing UTF-8 text and cluster mapping if the target surface does not use it.

## Since

1.8

**8.27.5.18 mark\_dirty() [1/2]**

```
void Cairo::Surface::mark_dirty ()
```

Tells cairo to consider the data buffer dirty.

In particular, if you've created an [ImageSurface](#) with a data buffer that you've allocated yourself and you draw to that data buffer using means other than cairo, you must call [mark\\_dirty\(\)](#) before doing any additional drawing to that surface with cairo.

Note that if you do draw to the [Surface](#) outside of cairo, you must call [flush\(\)](#) before doing the drawing.

**8.27.5.19 mark\_dirty() [2/2]**

```
void Cairo::Surface::mark_dirty (
    int x,
    int y,
    int width,
    int height)
```

Marks a rectangular area of the given surface dirty.

## Parameters

<i>x</i>	X coordinate of dirty rectangle
<i>y</i>	Y coordinate of dirty rectangle
<i>width</i>	width of dirty rectangle
<i>height</i>	height of dirty rectangle

**8.27.5.20 operator=()**

```
Surface & Cairo::Surface::operator= (
    const Surface & ) [delete]
```

**8.27.5.21 set\_device\_offset()**

```
void Cairo::Surface::set_device_offset (
    double x_offset,
    double y_offset)
```

Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.

One use case for this function is when we want to create a Surface that redirects drawing for a portion of an onscreen surface to an offscreen surface in a way that is completely invisible to the user of the cairo API. Setting a transformation via [Cairo::Context::translate\(\)](#) isn't sufficient to do this, since functions like [Cairo::Context::device\\_to\\_user\(\)](#) will expose the hidden offset.

Note that the offset only affects drawing to the surface, not using the surface in a surface pattern.

## Parameters

<i>x_offset</i>	the offset in the X direction, in device units
<i>y_offset</i>	the offset in the Y direction, in device units

**8.27.5.22 set\_device\_scale()** [1/2]

```
void Cairo::Surface::set_device_scale (
    double scale) [inline]
```

Sets x and y scale to the same value.

See [set\\_device\\_scale\(double, double\)](#) for details.

## Parameters

<i>scale</i>	a scale factor in the X and Y direction
--------------	-----------------------------------------

Since [cairomm 1.18](#)

### 8.27.5.23 set\_device\_scale() [2/2]

```
void Cairo::Surface::set_device_scale (
    double x_scale,
    double y_scale)
```

Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.

One common use for this is to render to very high resolution display devices at a scale factor, so that code that assumes 1 pixel will be a certain size will still work. Setting a transformation via [Cairo::Context::scale\(\)](#) isn't sufficient to do this, since functions like [Cairo::Context::device\\_to\\_user\(\)](#) will expose the hidden scale.

Note that the scale affects drawing to the surface as well as using the surface in a source pattern.

#### Parameters

<i>x_scale</i>	a scale factor in the X direction
<i>y_scale</i>	a scale factor in the Y direction

Since [cairomm 1.18](#)

### 8.27.5.24 set\_fallback\_resolution()

```
void Cairo::Surface::set_fallback_resolution (
    double x_pixels_per_inch,
    double y_pixels_per_inch)
```

Set the horizontal and vertical resolution for image fallbacks.

When certain operations aren't supported natively by a backend, cairo will fallback by rendering operations to an image and then overlaying that image onto the output. For backends that are natively vector-oriented, this function can be used to set the resolution used for these image fallbacks, (larger values will result in more detailed images, but also larger file sizes).

Some examples of natively vector-oriented backends are the ps, pdf, and svg backends.

For backends that are natively raster-oriented, image fallbacks are still possible, but they are always performed at the native device resolution. So this function has no effect on those backends.

Note: The fallback resolution only takes effect at the time of completing a page (with [Context::show\\_page\(\)](#) or [Context::copy\\_page\(\)](#)) so there is currently no way to have more than one fallback resolution in effect on a single page.

The default fallback resolution is 300 pixels per inch in both dimensions.

#### Parameters

<i>x_pixels_per_inch</i>	Pixels per inch in the x direction
<i>y_pixels_per_inch</i>	Pixels per inch in the y direction

Since

1.2

### 8.27.5.25 set\_mime\_data()

```
void Cairo::Surface::set_mime_data (
    const std::string & mime_type,
    unsigned char * data,
    unsigned long length,
    const SlotDestroy & slot_destroy)
```

Attach an image in the format mime\_type to surface.

To remove the data from a surface, call [unset\\_mime\\_data\(\)](#) with same mime type.

The attached image (or filename) data can later be used by backends which support it (currently: PDF, PS, SVG and Win32 Printing surfaces) to emit this data instead of making a snapshot of the surface. This approach tends to be faster and requires less memory and disk space.

The recognized MIME types are the following: CAIRO\_MIME\_TYPE\_JPEG, CAIRO\_MIME\_TYPE\_PNG, CAIRO\_MIME\_TYPE\_JP2, CAIRO\_MIME\_TYPE\_URI.

See corresponding backend surface docs for details about which MIME types it can handle. Caution: the associated MIME data will be discarded if you draw on the surface afterwards. Use this function with care.

#### Parameters

<i>mime_type</i>	The MIME type of the image data. param data The image @data to attach to the surface. param length The length of the image data.
<i>slot_destroy</i>	A callback slot that will be called when the <a href="#">Surface</a> no longer needs the data. For instance, when the <a href="#">Surface</a> is destroyed or when new image data is attached using the same MIME tpe.

Since

1.10

### 8.27.5.26 show\_page()

```
void Cairo::Surface::show_page ()
```

Emits and clears the current page for backends that support multiple pages.

Use [copy\\_page\(\)](#) if you don't want to clear the page.

Since

1.6

### 8.27.5.27 unset\_mime\_data()

```
void Cairo::Surface::unset_mime_data (
    const std::string & mime_type)
```

Remove the data from a surface.

See [set\\_mime\\_data\(\)](#).

### 8.27.5.28 write\_to\_png()

```
void Cairo::Surface::write_to_png (
    const std::string & filename)
```

Writes the contents of surface to a new file filename as a PNG image.

#### Note

For this function to be available, cairo must have been compiled with PNG support

#### Parameters

<i>filename</i>	the name of a file to write to
-----------------	--------------------------------

### 8.27.5.29 write\_to\_png\_stream()

```
void Cairo::Surface::write_to_png_stream (
    const SlotWriteFunc & write_func)
```

Writes the [Surface](#) to the write function.

#### Note

For this function to be available, cairo must have been compiled with PNG support

#### Parameters

<i>write_func</i>	The function to be called when the backend needs to write data to an output stream
-------------------	------------------------------------------------------------------------------------

#### Since

1.8

## 8.27.6 Member Data Documentation

### 8.27.6.1 m\_cobject

```
cobject* Cairo::Surface::m_cobject [protected]
```

The underlying C cairo surface type that is wrapped by this [Surface](#).

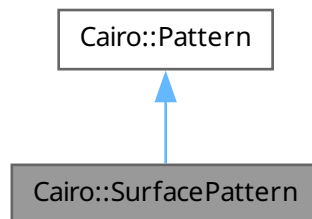
The documentation for this class was generated from the following file:

- caiomm/surface.h

## 8.28 Cairo::SurfacePattern Class Reference

```
#include <cairomm/pattern.h>
```

Inheritance diagram for Cairo::SurfacePattern:



### Public Types

- enum class [Filter](#) {  
[FAST](#) = CAIRO\_FILTER\_FAST ,  
[GOOD](#) = CAIRO\_FILTER\_GOOD ,  
[BEST](#) = CAIRO\_FILTER\_BEST ,  
[NEAREST](#) = CAIRO\_FILTER\_NEAREST ,  
[BILINEAR](#) = CAIRO\_FILTER\_BILINEAR ,  
[GAUSSIAN](#) = CAIRO\_FILTER\_GAUSSIAN }

*Filter is used to indicate what filtering should be applied when reading pixel values from patterns.*

### Public Types inherited from [Cairo::Pattern](#)

- enum class [Type](#) {  
[SOLID](#) = CAIRO\_PATTERN\_TYPE\_SOLID ,  
[SURFACE](#) = CAIRO\_PATTERN\_TYPE\_SURFACE ,  
[LINEAR](#) = CAIRO\_PATTERN\_TYPE\_LINEAR ,  
[RADIAL](#) = CAIRO\_PATTERN\_TYPE\_RADIAL }

*Type is used to describe the type of a given pattern.*

- enum class [Extend](#) {  
[NONE](#) = CAIRO\_EXTEND\_NONE ,  
[REPEAT](#) = CAIRO\_EXTEND\_REPEAT ,  
[REFLECT](#) = CAIRO\_EXTEND\_REFLECT ,  
[PAD](#) = CAIRO\_EXTEND\_PAD }

*Cairo::Extend is used to describe how pattern color/alpha will be determined for areas "outside" the pattern's natural area, (for example, outside the surface bounds or outside the gradient geometry).*

- typedef cairo\_pattern\_t [cobject](#)



**Public Member Functions**

- [SurfacePattern](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~SurfacePattern](#) () override
- void [set\\_filter](#) ([Filter](#) filter)  
*Sets the filter to be used for resizing when using this pattern.*
- [Filter](#) [get\\_filter](#) () const  
*Gets the current filter for a pattern.*
- [RefPtr](#)< const [Surface](#) > [get\\_surface](#) () const  
*Gets the surface associated with this pattern.*
- [RefPtr](#)< [Surface](#) > [get\\_surface](#) ()

**Public Member Functions inherited from Cairo::Pattern**

- [Pattern](#) (cairo\_pattern\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Pattern](#) (const [Pattern](#) &)=delete
- [Pattern](#) & [operator=](#) (const [Pattern](#) &)=delete
- virtual [~Pattern](#) ()
- void [set\\_matrix](#) (const [Matrix](#) &matrix)  
*Sets the pattern's transformation matrix to @matrix.*
- void [get\\_matrix](#) ([Matrix](#) &matrix) const  
*Returns the pattern's transformation matrix.*
- [Matrix](#) [get\\_matrix](#) () const  
*Returns the pattern's transformation matrix.*
- [Type](#) [get\\_type](#) () const  
*Returns the type of the pattern.*
- void [set\\_extend](#) ([Extend](#) extend)  
*Sets the mode to be used for drawing outside the area of a pattern.*
- [Extend](#) [get\\_extend](#) () const  
*Gets the current extend mode See Cairo::Extend for details on the semantics of each extend strategy.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

**Static Public Member Functions**

- static [RefPtr](#)< [SurfacePattern](#) > [create](#) (const [RefPtr](#)< [Surface](#) > &surface)  
*Create a new Cairo::Pattern for the given surface.*

**Protected Member Functions**

- [SurfacePattern](#) (const [RefPtr](#)< [Surface](#) > &surface)

## Protected Member Functions inherited from [Cairo::Pattern](#)

- [Pattern](#) ()

## Additional Inherited Members

## Protected Attributes inherited from [Cairo::Pattern](#)

- [cobject](#) \* [m\\_cobject](#)

## 8.28.1 Member Enumeration Documentation

### 8.28.1.1 Filter

```
enum class Cairo::SurfacePattern::Filter [strong]
```

Filter is used to indicate what filtering should be applied when reading pixel values from patterns.

See [Cairo::SurfacePattern::set\\_filter\(\)](#) for indicating the desired filter to be used with a particular pattern.

#### Enumerator

FAST	A high-performance filter, with quality similar to <a href="#">Cairo::Pattern::Filter::NEAREST</a> .
GOOD	A reasonable-performance filter, with quality similar to <a href="#">Cairo::BILINEAR</a> .
BEST	The highest-quality available, performance may not be suitable for interactive use.
NEAREST	Nearest-neighbor filtering.
BILINEAR	Linear interpolation in two dimensions.
GAUSSIAN	This filter value is currently unimplemented, and should not be used in current code.

## 8.28.2 Constructor & Destructor Documentation

### 8.28.2.1 [SurfacePattern\(\)](#) [1/2]

```
Cairo::SurfacePattern::SurfacePattern (  
    const RefPtr< Surface > & surface) [explicit], [protected]
```

### 8.28.2.2 [SurfacePattern\(\)](#) [2/2]

```
Cairo::SurfacePattern::SurfacePattern (  
    cairo\_pattern\_t * cobject,  
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a [RefPtr](#).

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	Whether we already have a reference. Otherwise, the constructor will take an extra reference.

**8.28.2.3 ~SurfacePattern()**

```
Cairo::SurfacePattern::~~SurfacePattern () [override]
```

**8.28.3 Member Function Documentation****8.28.3.1 create()**

```
static RefPtr< SurfacePattern > Cairo::SurfacePattern::create (
    const RefPtr< Surface > & surface) [static]
```

Create a new [Cairo::Pattern](#) for the given surface.

**8.28.3.2 get\_filter()**

```
Filter Cairo::SurfacePattern::get_filter () const
```

Gets the current filter for a pattern.

See [Cairo::Filter](#) for details on each filter.

**8.28.3.3 get\_surface() [1/2]**

```
RefPtr< Surface > Cairo::SurfacePattern::get_surface ()
```

**8.28.3.4 get\_surface() [2/2]**

```
RefPtr< const Surface > Cairo::SurfacePattern::get_surface () const
```

Gets the surface associated with this pattern.

Since

1.4

**8.28.3.5 set\_filter()**

```
void Cairo::SurfacePattern::set_filter (
    Filter filter)
```

Sets the filter to be used for resizing when using this pattern.

See [Cairo::Filter](#) for details on each filter.

Note that you might want to control filtering even when you do not have an explicit [Cairo::Pattern](#) object, (for example when using [Cairo::Context::set\\_source\\_surface\(\)](#)). In these cases, it is convenient to use [Cairo::Context::get\\_source\(\)](#) to get access to the pattern that cairo creates implicitly.

## Parameters

<i>filter</i>	Cairo::Filter describing the filter to use for resizing the pattern
---------------	---------------------------------------------------------------------

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

- cairomm/pattern.h

## 8.29 Cairo::SvgSurface Class Reference

A SvgSurface provides a way to render Scalable Vector Graphics (SVG) images from cairo.

```
#include <cairomm/surface.h>
```

Inheritance diagram for Cairo::SvgSurface:



### Public Member Functions

- [SvgSurface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~SvgSurface](#) () override
- void [restrict\\_to\\_version](#) (SvgVersion version)  
*Restricts the generated SVG file to the given version.*

### Public Member Functions inherited from [Cairo::Surface](#)

- [Surface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & [operator=](#) (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*

- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* **data**, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const  
*Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)  
*Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)  
*Writes the [Surface](#) to the write function.*
- [RefPtr](#)< [Device](#) > [get\\_device](#) ()  
*This function returns the device for a surface.*
- [cobject](#) \* [cobj](#) ()  
*Provides acces to the underlying C cairo surface.*
- const [cobject](#) \* [cobj](#) () const  
*Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr](#)< [SvgSurface](#) > [create](#) ( **std::string** filename, double width\_in\_points, double height\_in\_points)  
*Creates a [SvgSurface](#) with a specified dimensions that will be saved as the given filename.*
- static [RefPtr](#)< [SvgSurface](#) > [create\\_for\\_stream](#) (const [SlotWriteFunc](#) &write\_func, double width\_in\_points, double height\_in\_points)  
*Creates a [SvgSurface](#) with a specified dimensions that will be written to the given write function instead of saved directly to disk.*
- static const **std::vector**< [SvgVersion](#) > [get\\_versions](#) ()  
*Retrieves the list of SVG versions supported by cairo.*
- static **std::string** [version\\_to\\_string](#) ([SvgVersion](#) version)  
*Get the string representation of the given version id.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > other, [Content](#) content, int width, int height)  
*Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > &target, double x, double y, double width, double height)  
*Create a new surface that is a rectangle within the target surface.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {  
[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,  
[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,  
[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,  
[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,  
[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,  
[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,  
[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,  
[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,  
[WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,  
[QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,  
[SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,  
[QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,  
[RECORDING](#) = CAIRO\_SURFACE\_TYPE\_RECORDING ,  
[VG](#) = CAIRO\_SURFACE\_TYPE\_VG ,  
[GL](#) = CAIRO\_SURFACE\_TYPE\_GL ,  
[DRM](#) = CAIRO\_SURFACE\_TYPE\_DRM ,  
[TEE](#) = CAIRO\_SURFACE\_TYPE\_TEE ,  
[XML](#) = CAIRO\_SURFACE\_TYPE\_XML ,  
[SKIA](#) = CAIRO\_SURFACE\_TYPE\_SKIA ,  
[SUBSURFACE](#) = CAIRO\_SURFACE\_TYPE\_SUBSURFACE }

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }  
*Format is used to identify the memory format of image data.*
- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)  
*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*
- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)  
*This is the type of function which is called when a backend needs to read data from an input stream.*
- typedef sigc::slot< void()> [SlotDestroy](#)  
*For instance, void on\_destroy();.*
- typedef cairo\_surface\_t [cobject](#)  
*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)  
*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.29.1 Detailed Description

A [SvgSurface](#) provides a way to render Scalable Vector Graphics (SVG) images from cairo.

This surface is not rendered to the screen but instead renders the drawing to an SVG file on disk.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with SVG support

## 8.29.2 Constructor & Destructor Documentation

### 8.29.2.1 [SvgSurface\(\)](#)

```
Cairo::SvgSurface::SvgSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a [RefPtr](#).

#### Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

### 8.29.2.2 ~SvgSurface()

```
Cairo::SvgSurface::~SvgSurface () [override]
```

## 8.29.3 Member Function Documentation

### 8.29.3.1 create()

```
static RefPtr< SvgSurface > Cairo::SvgSurface::create (
    std::string filename,
    double width_in_points,
    double height_in_points) [static]
```

Creates a [SvgSurface](#) with a specified dimensions that will be saved as the given filename.

#### Parameters

<i>filename</i>	The name of the SVG file to save the surface to
<i>width_in_points</i>	The width of the SVG document in points
<i>height_in_points</i>	The height of the SVG document in points

#### Examples

[svg-surface.cc](#).

### 8.29.3.2 create\_for\_stream()

```
static RefPtr< SvgSurface > Cairo::SvgSurface::create_for_stream (
    const SlotWriteFunc & write_func,
    double width_in_points,
    double height_in_points) [static]
```

Creates a [SvgSurface](#) with a specified dimensions that will be written to the given write function instead of saved directly to disk.

#### Parameters

<i>write_func</i>	The function to be called when the backend needs to write data to an output stream
<i>width_in_points</i>	The width of the SVG document in points
<i>height_in_points</i>	The height of the SVG document in points

#### Since

1.8



### 8.29.3.3 get\_versions()

```
static const std::vector< SvgVersion > Cairo::SvgSurface::get_versions () [static]
```

Retrieves the list of SVG versions supported by cairo.

See [restrict\\_to\\_version\(\)](#).

Since

1.2

### 8.29.3.4 restrict\_to\_version()

```
void Cairo::SvgSurface::restrict_to_version (  
    SvgVersion version)
```

Restricts the generated SVG file to the given version.

See [get\\_versions\(\)](#) for a list of available version values that can be used here.

This function should only be called before any drawing operations have been performed on the given surface. The simplest way to do this is to call this function immediately after creating the surface.

Since

1.2

### 8.29.3.5 version\_to\_string()

```
static std::string Cairo::SvgSurface::version_to_string (  
    SvgVersion version) [static]
```

Get the string representation of the given version id.

The returned string will be empty if version isn't valid. See [get\\_versions\(\)](#) for a way to get the list of valid version ids.

since: 1.2

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

- caiomm/surface.h

## 8.30 Cairo::ToyFontFace Class Reference

A simple font face used for the cairo 'toy' font API.

```
#include <cairomm/fontface.h>
```

Inheritance diagram for Cairo::ToyFontFace:



### Public Types

- enum class [Slant](#) {  
    [NORMAL](#) = CAIRO\_FONT\_SLANT\_NORMAL ,  
    [ITALIC](#) = CAIRO\_FONT\_SLANT\_ITALIC ,  
    [OBLIQUE](#) = CAIRO\_FONT\_SLANT\_OBLIQUE }  
    *Specifies variants of a font face based on their slant.*
- enum class [Weight](#) {  
    [NORMAL](#) = CAIRO\_FONT\_WEIGHT\_NORMAL ,  
    [BOLD](#) = CAIRO\_FONT\_WEIGHT\_BOLD }  
    *Specifies variants of a font face based on their weight.*

### Public Types inherited from [Cairo::FontFace](#)

- typedef cairo\_font\_face\_t [cobject](#)

### Public Member Functions

- **std::string** [get\\_family](#) () const  
    *Gets the family name of a toy font.*
- [Slant](#) [get\\_slant](#) () const  
    *Gets the slant a toy font.*
- [Weight](#) [get\\_weight](#) () const  
    *Gets the weight a toy font.*

## Public Member Functions inherited from Cairo::FontFace

- [FontFace](#) (cairo\_font\_face\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [FontFace](#) (const [FontFace](#) &)=delete
- [FontFace](#) & [operator=](#) (const [FontFace](#) &)=delete
- virtual [~FontFace](#) ()
- [FontType](#) [get\\_type](#) () const  
*Returns the type of the backend used to create a font face.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Static Public Member Functions

- static [RefPtr](#)< [ToyFontFace](#) > [create](#) (const **std::string** &family, [Slant](#) slant, [Weight](#) weight)  
*Creates a font face from a triplet of family, slant, and weight.*

## Protected Member Functions

- [ToyFontFace](#) (const **std::string** &family, [Slant](#) slant, [Weight](#) weight)

## Additional Inherited Members

## Protected Attributes inherited from Cairo::FontFace

- [cobject](#) \* [m\\_cobject](#)

### 8.30.1 Detailed Description

A simple font face used for the cairo 'toy' font API.

Since

1.8

### 8.30.2 Member Enumeration Documentation

#### 8.30.2.1 Slant

```
enum class Cairo::ToyFontFace::Slant [strong]
```

Specifies variants of a font face based on their slant.

**Enumerator**

NORMAL	Upright font style.
ITALIC	Italic font style.
OBLIQUE	Oblique font style.

**8.30.2.2 Weight**

```
enum class Cairo::ToyFontFace::Weight [strong]
```

Specifies variants of a font face based on their weight.

**Enumerator**

NORMAL	Normal font weight.
BOLD	Bold font weight.

**8.30.3 Constructor & Destructor Documentation****8.30.3.1 ToyFontFace()**

```
Cairo::ToyFontFace::ToyFontFace (
    const std::string & family,
    Slant slant,
    Weight weight) [protected]
```

**8.30.4 Member Function Documentation****8.30.4.1 create()**

```
static RefPtr< ToyFontFace > Cairo::ToyFontFace::create (
    const std::string & family,
    Slant slant,
    Weight weight) [static]
```

Creates a font face from a triplet of family, slant, and weight.

These font faces are used in implementation of the the [Context](#) "toy" font API.

If family is the zero-length string "", the platform-specific default family is assumed. The default family then can be queried using [get\\_family\(\)](#).

The [Context::select\\_font\\_face\(\)](#) function uses this to create font faces. See that function for limitations of toy font faces.

**Parameters**

<i>family</i>	a font family name, encoded in UTF-8.
<i>slant</i>	the slant for the font.
<i>weight</i>	the weight for the font.

**Examples**

[toy-text.cc](#), and [user-font.cc](#).

### 8.30.4.2 get\_family()

```
std::string Cairo::ToyFontFace::get_family () const
```

Gets the family name of a toy font.

### 8.30.4.3 get\_slant()

```
Slant Cairo::ToyFontFace::get_slant () const
```

Gets the slant a toy font.

### 8.30.4.4 get\_weight()

```
Weight Cairo::ToyFontFace::get_weight () const
```

Gets the weight a toy font.

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

- cairomm/fontface.h

## 8.31 Cairo::UserFontFace Class Reference

Font support with font data provided by the user.

```
#include <cairomm/fontface.h>
```

Inheritance diagram for Cairo::UserFontFace:



### Public Member Functions

- [~UserFontFace](#) () override

## Public Member Functions inherited from [Cairo::FontFace](#)

- [FontFace](#) (cairo\_font\_face\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [FontFace](#) (const [FontFace](#) &)=delete
- [FontFace](#) & [operator=](#) (const [FontFace](#) &)=delete
- virtual [~FontFace](#) ()
- [FontType](#) [get\\_type](#) () const  
*Returns the type of the backend used to create a font face.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Protected Member Functions

- virtual [ErrorStatus](#) [init](#) (const [RefPtr](#)< [ScaledFont](#) > &scaled\_font, const [RefPtr](#)< [Context](#) > &cr, [FontExtents](#) &extents)  
*This function is called when a scaled-font needs to be created for a user font-face.*
- virtual [ErrorStatus](#) [unicode\\_to\\_glyph](#) (const [RefPtr](#)< [ScaledFont](#) > &scaled\_font, unsigned long unicode, unsigned long &glyph)  
*This function is called to convert an input Unicode character to a single glyph.*
- virtual [ErrorStatus](#) [render\\_glyph](#) (const [RefPtr](#)< [ScaledFont](#) > &scaled\_font, unsigned long glyph, const [RefPtr](#)< [Context](#) > &cr, [TextExtents](#) &metrics)=0  
*This function is called when a user scaled-font needs to render a glyph.*
- virtual [ErrorStatus](#) [text\\_to\\_glyphs](#) (const [RefPtr](#)< [ScaledFont](#) > &scaled\_font, const [std::string](#) &utf8, [std::vector](#)< [Glyph](#) > &glyphs, [std::vector](#)< [TextCluster](#) > &clusters, [TextClusterFlags](#) &cluster\_flags)  
*This function is called to convert input text to an array of glyphs.*
- [UserFontFace](#) ()

## Additional Inherited Members

## Public Types inherited from [Cairo::FontFace](#)

- typedef cairo\_font\_face\_t [cobject](#)

## Protected Attributes inherited from [Cairo::FontFace](#)

- [cobject](#) \* [m\\_cobject](#)

### 8.31.1 Detailed Description

Font support with font data provided by the user.

The user-font feature allows the cairo user to provide drawings for glyphs in a font. This is most useful in implementing fonts in non-standard formats, like SVG fonts and Flash fonts, but can also be used by games and other application to draw "funky" fonts.

To use user fonts, you must derive from this class and implement the virtual functions below. The only virtual function that absolutely must be implemented is `render_glyph()`. You should make the constructor protected and provide a factory function that returns a new object in a `RefPtr` since it is a refcounted object

```
class MyUserFont : public UserFontFace {
public:
    static Cairo::RefPtr<MyUserFont> create() {
        return Cairo::make_refptr_for_instance<MyUserFont>(new MyUserFont);
    }
protected:
    // implement render_glyph() and any other virtual functions you want to override
    ErrorStatus render_glyph(const RefPtr<ScaledFont>& scaled_font,
                           unsigned long glyph,
                           const RefPtr<Context>& cr,
                           TextExtents& metrics) {
        // render the glyph into cr here
    }

    MyUserFont() : UserFontFace() {
        // constructor implementation
    }
};
```

#### Warning

Because of a design flaw in cairomm, it is currently necessary to keep the the `UserFontFace` object around until as long as you are rendering text with the user font. The following code illustrates the issue:

```
{
    auto face = MyUserFont::create();
    cr->set_font_face(face);
} // scope for demonstration purposes

// the following call will cause a crash because your user font is no longer
// in scope but it needs to call the virtual functions in face
cr->show_text("hello, world");
```

The preceding is obviously a very contrived example, but the important thing to know is that you *must* cache all userfont objects yourself as long as you intend to render text with that font. A future release of cairomm will fix this requirement, but that will require ABI-incompatible changes.

#### Since

1.8

#### Examples

[user-font.cc](http://user-font.cc).

### 8.31.2 Constructor & Destructor Documentation

#### 8.31.2.1 ~UserFontFace()

```
Cairo::UserFontFace::~UserFontFace () [override]
```

### 8.31.2.2 UserFontFace()

```
Cairo::UserFontFace::UserFontFace () [protected]
```

## 8.31.3 Member Function Documentation

### 8.31.3.1 init()

```
virtual ErrorStatus Cairo::UserFontFace::init (  
    const RefPtr< ScaledFont > & scaled_font,  
    const RefPtr< Context > & cr,  
    FontExtents & extents) [protected], [virtual]
```

This function is called when a scaled-font needs to be created for a user font-face.

The [Context](#) *cr* is not used by the caller, but is prepared in font space, similar to what the cairo contexts passed to the `render_glyph` method will look like. The callback can use this context for extents computation for example. After the callback is called, *cr* is checked for any error status.

The *extents* argument is where the user font sets the font extents for *scaled\_font*. It is in font space, which means that for most cases its ascent and descent members should add to 1.0. *extents* is preset to hold a value of 1.0 for ascent, height, and `max_x_advance`, and 0.0 for descent and `max_y_advance` members.

The default implementation sets the font extents as described in the previous paragraph. If you need different extents, you can override this function in your derived class.

Note that *scaled\_font* is not fully initialized at this point and trying to use it for text operations in the callback will result in deadlock.

#### Parameters

<i>scaled_font</i>	the scaled-font being created
<i>cr</i>	cairo context, in font space
<i>extents</i>	extents to fill in, in font space

#### Returns

CAIRO\_STATUS\_SUCCESS upon success, or CAIRO\_STATUS\_USER\_FONT\_ERROR or any other error status on error.

#### Since

1.8

#### Examples

[user-font.cc](#).



### 8.31.3.2 render\_glyph()

```
virtual ErrorStatus Cairo::UserFontFace::render_glyph (
    const RefPtr< ScaledFont > & scaled_font,
    unsigned long glyph,
    const RefPtr< Context > & cr,
    TextExtents & metrics) [protected], [pure virtual]
```

This function is called when a user scaled-font needs to render a glyph.

You must implement this in your derived class, and it is expected to draw the glyph with code *glyph* to the [Context](#) *cr*. *cr* is prepared such that the glyph drawing is done in font space. That is, the matrix set on *cr* is the scale matrix of *scaled\_font*. The *extents* argument is where the user font sets the font extents for *scaled\_font*. However, if user prefers to draw in user space, they can achieve that by changing the matrix on *cr*. All cairo rendering operations to *cr* are permitted, however, the result is undefined if any source other than the default source on *cr* is used. That means, glyph bitmaps should be rendered using [Context::mask\(\)](#) instead of [Context::paint\(\)](#).

Other non-default settings on *cr* include a font size of 1.0 (given that it is set up to be in font space), and font options corresponding to *scaled\_font*.

The *extents* argument is preset to have *x\_bearing*, *width*, and *y\_advance* of zero, *y\_bearing* set to *-font\_extents.ascent*, *height* to *font\_extents.ascent+font\_extents.descent*, and *x\_advance* to *font\_extents.max\_x\_advance*. The only field user needs to set in majority of cases is *x\_advance*. If the *width* field is zero upon this function returning (which is its preset value), the glyph extents are automatically computed based on the drawings done to *cr*. This is in most cases exactly what the desired behavior is. However, if for any reason this function sets the extents, it must be ink extents, and include the extents of all drawing done to *cr*.

#### Parameters

<i>scaled_font</i>	user scaled-font
<i>glyph</i>	glyph code to render
<i>cr</i>	<a href="#">Context</a> to draw to, in font space
<i>extents</i>	glyph extents to fill in, in font space

#### Returns

CAIRO\_STATUS\_SUCCESS upon success, or CAIRO\_STATUS\_USER\_FONT\_ERROR or any other error status on error.

#### Since

1.8

#### Examples

[user-font.cc](http://user-font.cc).

### 8.31.3.3 text\_to\_glyphs()

```
virtual ErrorStatus Cairo::UserFontFace::text_to_glyphs (
    const RefPtr< ScaledFont > & scaled_font,
    const std::string & utf8,
    std::vector< Glyph > & glyphs,
    std::vector< TextCluster > & clusters,
    TextClusterFlags & cluster_flags) [protected], [virtual]
```

This function is called to convert input text to an array of glyphs.

This is used by the [Context::show\\_text\(\)](#) operation.

Using this function, the user-font has full control on glyphs and their positions. That means, it allows for features like ligatures and kerning, as well as complex shaping required for scripts like Arabic and Indic.

This function should populate the glyph indices and positions (in font space) assuming that the text is to be shown at the origin.

If clusters is not empty, cluster mapping should be computed.

If you do not override this virtual function in your derived class, the `unicode_to_glyph` function is used instead.

Note: While cairo does not impose any limitation on glyph indices, some applications may assume that a glyph index fits in a 16-bit unsigned integer. As such, it is advised that user-fonts keep their glyphs in the 0 to 65535 range. Furthermore, some applications may assume that glyph 0 is a special glyph-not-found glyph. User-fonts are advised to use glyph 0 for such purposes and do not use that glyph value for other purposes.

#### Parameters

<i>scaled_font</i>	the scaled-font being created
<i>utf8</i>	a string of text encoded in UTF-8
<i>glyphs</i>	array of glyphs to fill, in font space
<i>clusters</i>	array of cluster mapping information to fill
<i>cluster_flags</i>	a variable to set the cluster flags corresponding to the output clusters

#### Returns

CAIRO\_STATUS\_SUCCESS upon success, or CAIRO\_STATUS\_USER\_FONT\_ERROR or any other error status on error.

#### Since

1.8

### 8.31.3.4 unicode\_to\_glyph()

```
virtual ErrorStatus Cairo::UserFontFace::unicode_to_glyph (
    const RefPtr< ScaledFont > & scaled_font,
    unsigned long unicode,
    unsigned long & glyph) [protected], [virtual]
```

This function is called to convert an input Unicode character to a single glyph.

This is used by the [Context::show\\_text\(\)](#) operation.

This function is used to provide the same functionality as the `text_to_glyphs` callback does but has much less control on the output, in exchange for increased ease of use. The inherent assumption to using this callback is that each character maps to one glyph, and that the mapping is context independent. It also assumes that glyphs are positioned according to their advance width. These mean no ligatures, kerning, or complex scripts can be implemented using this callback.

The default implementation of this function is an identity mapping from Unicode code-points to glyph indices. If you need different behavior, you may override this virtual function in your derived class.

Note: While cairo does not impose any limitation on glyph indices, some applications may assume that a glyph index fits in a 16-bit unsigned integer. As such, it is advised that user-fonts keep their glyphs in the 0 to 65535 range. Furthermore, some applications may assume that glyph 0 is a special glyph-not-found glyph. User-fonts are advised to use glyph 0 for such purposes and do not use that glyph value for other purposes.

#### Parameters

<i>scaled_font</i>	the scaled-font being created
<i>unicode</i>	input unicode character code-point
<i>glyph_index</i>	output glyph index

#### Returns

CAIRO\_STATUS\_SUCCESS upon success, or CAIRO\_STATUS\_USER\_FONT\_ERROR or any other error status on error.

#### Since

1.8

#### Examples

[user-font.cc](http://user-font.cc).

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

- `caiomm/fontface.h`

## 8.32 Cairo::Win32FontFace Class Reference

Font support for Microsoft Windows.

```
#include <cairomm/win32_font.h>
```

Inheritance diagram for Cairo::Win32FontFace:



### Static Public Member Functions

- static [RefPtr< Win32FontFace > create](#) (LOGFONTW \*logfont)  
*Creates a new font for the Win32 font backend based on a LOGFONT.*
- static [RefPtr< Win32FontFace > create](#) (HFONT font)  
*Creates a new font for the Win32 font backend based on a HFONT.*
- static [RefPtr< Win32FontFace > create](#) (LOGFONTW \*logfont, HFONT font)  
*Creates a new font for the Win32 font backend based on a LOGFONT.*

### Protected Member Functions

- [Win32FontFace](#) (LOGFONTW \*logfont)
- [Win32FontFace](#) (HFONT font)
- [Win32FontFace](#) (LOGFONTW \*logfont, HFONT font)

### Additional Inherited Members

### Public Types inherited from [Cairo::FontFace](#)

- typedef cairo\_font\_face\_t [cobject](#)

## Public Member Functions inherited from Cairo::FontFace

- [FontFace](#) (cairo\_font\_face\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [FontFace](#) (const [FontFace](#) &)=delete
- [FontFace](#) & [operator=](#) (const [FontFace](#) &)=delete
- virtual [~FontFace](#) ()
- [FontType](#) [get\\_type](#) () const  
*Returns the type of the backend used to create a font face.*
- [cobject](#) \* [cobj](#) ()
- const [cobject](#) \* [cobj](#) () const
- void [reference](#) () const
- void [unreference](#) () const

## Protected Attributes inherited from Cairo::FontFace

- [cobject](#) \* [m\\_cobject](#)

### 8.32.1 Detailed Description

Font support for Microsoft Windows.

Since

1.8

### 8.32.2 Constructor & Destructor Documentation

#### 8.32.2.1 Win32FontFace() [1/3]

```
Cairo::Win32FontFace::Win32FontFace (
    LOGFONTW * logfont) [protected]
```

#### 8.32.2.2 Win32FontFace() [2/3]

```
Cairo::Win32FontFace::Win32FontFace (
    HFONT font) [protected]
```

#### 8.32.2.3 Win32FontFace() [3/3]

```
Cairo::Win32FontFace::Win32FontFace (
    LOGFONTW * logfont,
    HFONT font) [protected]
```

### 8.32.3 Member Function Documentation

#### 8.32.3.1 create() [1/3]

```
static RefPtr< Win32FontFace > Cairo::Win32FontFace::create (
    HFONT font) [static]
```

Creates a new font for the Win32 font backend based on a HFONT.

This font can then be used with [Context::set\\_font\\_face\(\)](#) or [Win32ScaledFont::create\(\)](#).

## Parameters

<i>font</i>	An HFONT structure specifying the font to use.
-------------	------------------------------------------------

## Since

1.8

**8.32.3.2 create() [2/3]**

```
static RefPtr< Win32FontFace > Cairo::Win32FontFace::create (
    LOGFONTW * logfont) [static]
```

Creates a new font for the Win32 font backend based on a LOGFONT.

This font can then be used with [Context::set\\_font\\_face\(\)](#) or [Win32ScaledFont::create\(\)](#).

## Parameters

<i>logfont</i>	A LOGFONTW structure specifying the font to use. The lfHeight, lfWidth, lfOrientation and lfEscapement fields of this structure are ignored.
----------------	----------------------------------------------------------------------------------------------------------------------------------------------

## Since

1.8

**8.32.3.3 create() [3/3]**

```
static RefPtr< Win32FontFace > Cairo::Win32FontFace::create (
    LOGFONTW * logfont,
    HFONT font) [static]
```

Creates a new font for the Win32 font backend based on a LOGFONT.

This font can then be used with [Context::set\\_font\\_face\(\)](#) or [Win32ScaledFont::create\(\)](#).

## Parameters

<i>logfont</i>	A LOGFONTW structure specifying the font to use. If hfont is null then the lfHeight, lfWidth, lfOrientation and lfEscapement fields of this structure are ignored. Otherwise lfWidth, lfOrientation and lfEscapement must be zero.
<i>font</i>	An HFONT that can be used when the font matrix is a scale by -lfHeight and the CTM is identity.

## Since

1.8

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

- `cairomm/win32_font.h`

## 8.33 Cairo::Win32PrintingSurface Class Reference

A multi-page vector surface type for printing on Microsoft Windows.

```
#include <cairomm/win32_surface.h>
```

Inheritance diagram for Cairo::Win32PrintingSurface:



### Public Member Functions

- [Win32PrintingSurface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)
- [~Win32PrintingSurface](#) () override

### Public Member Functions inherited from [Cairo::Surface](#)

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & operator= (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* data, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)

- Marks a rectangular area of the given surface dirty.*

  - void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)

*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*

- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const

*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*

- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)

*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*

- void [set\\_device\\_scale](#) (double scale)

*Sets x and y scale to the same value.*

- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const

*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*

- double [get\\_device\\_scale](#) () const

*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*

- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)

*Set the horizontal and vertical resolution for image fallbacks.*

- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const

*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*

- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const

*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*

- void [copy\\_page](#) ()

*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*

- void [show\\_page](#) ()

*Emits and clears the current page for backends that support multiple pages.*

- bool [has\\_show\\_text\\_glyphs](#) () const

*Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*

- void [write\\_to\\_png](#) (const **std::string** &filename)

*Writes the contents of surface to a new file filename as a PNG image.*

- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)

*Writes the [Surface](#) to the write function.*

- [RefPtr](#)< [Device](#) > [get\\_device](#) ()

*This function returns the device for a surface.*

- [cobject](#) \* [cobj](#) ()

*Provides acces to the underlying C cairo surface.*

- const [cobject](#) \* [cobj](#) () const

*Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr](#)< [Win32PrintingSurface](#) > [create](#) (HDC hdc)

*Creates a cairo surface that targets the given DC.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > other, [Content](#) content, int width, int height)

*Create a new surface that is as compatible as possible with an existing surface.*

- static [RefPtr](#)< [Surface](#) > [create](#) (const [RefPtr](#)< [Surface](#) > &target, double x, double y, double width, double height)

*Create a new surface that is a rectangle within the target surface.*



## Additional Inherited Members

### Public Types inherited from Cairo::Surface

- enum class [Type](#) {  
[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,  
[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,  
[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,  
[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,  
[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,  
[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,  
[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,  
[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,  
[WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,  
[QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,  
[SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,  
[QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,  
[RECORDING](#) = CAIRO\_SURFACE\_TYPE\_RECORDING ,  
[VG](#) = CAIRO\_SURFACE\_TYPE\_VG ,  
[GL](#) = CAIRO\_SURFACE\_TYPE\_GL ,  
[DRM](#) = CAIRO\_SURFACE\_TYPE\_DRM ,  
[TEE](#) = CAIRO\_SURFACE\_TYPE\_TEE ,  
[XML](#) = CAIRO\_SURFACE\_TYPE\_XML ,  
[SKIA](#) = CAIRO\_SURFACE\_TYPE\_SKIA ,  
[SUBSURFACE](#) = CAIRO\_SURFACE\_TYPE\_SUBSURFACE }

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)

*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*

- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)

*This is the type of function which is called when a backend needs to read data from an input stream.*

- typedef sigc::slot< void()> [SlotDestroy](#)

*For instance, `void on_destroy();`.*

- typedef cairo\_surface\_t [cobject](#)

*The underlying C cairo surface type.*

### Protected Attributes inherited from Cairo::Surface

- [cobject](#) \* [m\\_cobject](#)

*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.33.1 Detailed Description

A multi-page vector surface type for printing on Microsoft Windows.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with Win32 support

#### Since

1.8

### 8.33.2 Constructor & Destructor Documentation

#### 8.33.2.1 Win32PrintingSurface()

```
Cairo::Win32PrintingSurface::Win32PrintingSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]
```

#### 8.33.2.2 ~Win32PrintingSurface()

```
Cairo::Win32PrintingSurface::~~Win32PrintingSurface () [override]
```

### 8.33.3 Member Function Documentation

#### 8.33.3.1 create()

```
static RefPtr< Win32PrintingSurface > Cairo::Win32PrintingSurface::create (
    HDC hdc) [static]
```

Creates a cairo surface that targets the given DC.

The DC will be queried for its initial clip extents, and this will be used as the size of the cairo surface. The DC should be a printing DC; antialiasing will be ignored, and GDI will be used as much as possible to draw to the surface.

The returned surface will be wrapped using the paginated surface to provide correct complex rendering behaviour; [show\\_page\(\)](#) and associated methods must be used for correct output.

#### Parameters

<i>hdc</i>	the DC to create a surface for.
------------	---------------------------------

#### Since

1.8

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

- `caiomm/win32_surface.h`

## 8.34 Cairo::Win32ScaledFont Class Reference

Scaled Font implementation for Microsoft Windows fonts.

```
#include <cairomm/win32_font.h>
```

Inheritance diagram for Cairo::Win32ScaledFont:



### Public Member Functions

- void [select\\_font](#) (HDC hdc)  
*Selects the font into the given device context and changes the map mode and world transformation of the device context to match that of the font.*
- void [done\\_font](#) ()  
*Releases any resources allocated by [select\\_font\(\)](#)*
- double [get\\_metrics\\_factor](#) () const  
*Gets a scale factor between logical coordinates in the coordinate space used by [select\\_font\(\)](#) (that is, the coordinate system used by the Windows functions to return metrics) and font space coordinates.*
- void [get\\_logical\\_to\\_device](#) (Matrix &logical\_to\_device) const  
*Gets the transformation mapping the logical space used by this scaled font to device space.*
- void [get\\_device\\_to\\_logical](#) (Matrix &device\_to\_logical) const  
*Gets the transformation mapping device space to the logical space used by this scaled font.*

### Public Member Functions inherited from Cairo::ScaledFont

- [cobject](#) \* [cobj](#) ()  
*Provides acces to the underlying C cairo object.*
- const [cobject](#) \* [cobj](#) () const  
*Provides acces to the underlying C cairo object.*
- [ScaledFont](#) ([cobject](#) \*[cobj](#), bool has\_reference=false)  
*Create a C++ wrapper object from the C instance.*
- [ScaledFont](#) (const [ScaledFont](#) &)=delete
- [ScaledFont](#) & [operator=](#) (const [ScaledFont](#) &)=delete
- virtual [~ScaledFont](#) ()
- void [get\\_extents](#) ([FontExtents](#) &extents) const  
*Gets the metrics for a [ScaledFont](#).*
- void [get\\_text\\_extents](#) (const [std::string](#) &utf8, [TextExtents](#) &extents) const

*Gets the extents for a string of text.*

- void [get\\_glyph\\_extents](#) (const **std::vector**< [Glyph](#) > &glyphs, [TextExtents](#) &extents) const

*Gets the extents for an array of glyphs.*

- [RefPtr](#)< [FontFace](#) > [get\\_font\\_face](#) () const

*The [FontFace](#) with which this [ScaledFont](#) was created.*

- void [get\\_font\\_options](#) ([FontOptions](#) &options) const

*Gets the [FontOptions](#) with which the [ScaledFont](#) was created.*

- void [get\\_font\\_matrix](#) ([Matrix](#) &font\_matrix) const

*Gets the font matrix with which the [ScaledFont](#) was created.*

- void [get\\_ctm](#) ([Matrix](#) &ctm) const

*Gets the CTM with which the [ScaledFont](#) was created.*

- [FontType](#) [get\\_type](#) () const

*Gets the type of scaled Font.*

- void [text\\_to\\_glyphs](#) (double x, double y, const **std::string** &utf8, **std::vector**< [Glyph](#) > &glyphs, **std::vector**< [TextCluster](#) > &clusters, [TextClusterFlags](#) &cluster\_flags)

- void [get\\_scale\\_matrix](#) ([Matrix](#) &scale\_matrix) const

*Stores the scale matrix of this scaled font into matrix.*

### Static Public Member Functions

- static [RefPtr](#)< [Win32ScaledFont](#) > [create](#) (const [RefPtr](#)< [Win32FontFace](#) > &font\_face, const [Matrix](#) &font\_matrix, const [Matrix](#) &ctm, const [FontOptions](#) &options=[FontOptions](#)())

*Creates a scaled font for the given [Win32FontFace](#).*

### Static Public Member Functions inherited from [Cairo::ScaledFont](#)

- static [RefPtr](#)< [ScaledFont](#) > [create](#) (const [RefPtr](#)< [FontFace](#) > &font\_face, const [Matrix](#) &font\_matrix, const [Matrix](#) &ctm, const [FontOptions](#) &options=[FontOptions](#)())

*Creates a [ScaledFont](#) object from a font face and matrices that describe the size of the font and the environment in which it will be used.*

### Protected Member Functions

- [Win32ScaledFont](#) (const [RefPtr](#)< [Win32FontFace](#) > &font\_face, const [Matrix](#) &font\_matrix, const [Matrix](#) &ctm, const [FontOptions](#) &options=[FontOptions](#)())

### Protected Member Functions inherited from [Cairo::ScaledFont](#)

- [ScaledFont](#) (const [RefPtr](#)< [FontFace](#) > &font\_face, const [Matrix](#) &font\_matrix, const [Matrix](#) &ctm, const [FontOptions](#) &options=[FontOptions](#)())

### Additional Inherited Members

### Public Types inherited from [Cairo::ScaledFont](#)

- typedef cairo\_scaled\_font\_t [cobject](#)

*The underlying C cairo object type.*

## Protected Attributes inherited from Cairo::ScaledFont

- `cobject * m_cobject`

The underlying C cairo object that is wrapped by this [ScaledFont](#).

### 8.34.1 Detailed Description

Scaled Font implementation for Microsoft Windows fonts.

Since

1.8

### 8.34.2 Constructor & Destructor Documentation

#### 8.34.2.1 Win32ScaledFont()

```
Cairo::Win32ScaledFont::Win32ScaledFont (
    const RefPtr< Win32FontFace > & font_face,
    const Matrix & font_matrix,
    const Matrix & ctm,
    const FontOptions & options = FontOptions()) [protected]
```

### 8.34.3 Member Function Documentation

#### 8.34.3.1 create()

```
static RefPtr< Win32ScaledFont > Cairo::Win32ScaledFont::create (
    const RefPtr< Win32FontFace > & font_face,
    const Matrix & font_matrix,
    const Matrix & ctm,
    const FontOptions & options = FontOptions()) [static]
```

Creates a scaled font for the given [Win32FontFace](#).

Since

1.8

#### 8.34.3.2 done\_font()

```
void Cairo::Win32ScaledFont::done_font ()
```

Releases any resources allocated by [select\\_font\(\)](#)

Since

1.8

#### 8.34.3.3 get\_device\_to\_logical()

```
void Cairo::Win32ScaledFont::get_device_to_logical (
    Matrix & device_to_logical) const
```

Gets the transformation mapping device space to the logical space used by this scaled font.

## Parameters

<i>device_to_logical</i>	matrix to return
--------------------------	------------------

## Since

1.8

**8.34.3.4 get\_logical\_to\_device()**

```
void Cairo::Win32ScaledFont::get_logical_to_device (
    Matrix & logical_to_device) const
```

Gets the transformation mapping the logical space used by this scaled font to device space.

## Parameters

<i>logical_to_device</i>	matrix to return
--------------------------	------------------

## Since

1.8

**8.34.3.5 get\_metrics\_factor()**

```
double Cairo::Win32ScaledFont::get_metrics_factor () const
```

Gets a scale factor between logical coordinates in the coordinate space used by [select\\_font\(\)](#) (that is, the coordinate system used by the Windows functions to return metrics) and font space coordinates.

## Returns

factor to multiply logical units by to get font space coordinates.

## Since

1.8

**8.34.3.6 select\_font()**

```
void Cairo::Win32ScaledFont::select_font (
    HDC hdc)
```

Selects the font into the given device context and changes the map mode and world transformation of the device context to match that of the font.

This function is intended for use when using layout APIs such as Uniscribe to do text layout with the cairo font. After finishing using the device context, you must call [done\\_font\(\)](#) to release any resources allocated by this function.

See [get\\_metrics\\_factor\(\)](#) for converting logical coordinates from the device context to font space.

Normally, calls to `SaveDC()` and `RestoreDC()` would be made around the use of this function to preserve the original graphics state.

## Parameters

<i>scaled_font</i>	A <code>cairo_scaled_font_t</code> from the Win32 font backend. Such an object can be created with <code>Win32FontFace::create_for_logfontw()</code> .
<i>hdc</i>	a device context

## Since

1.8

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

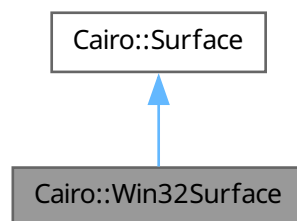
- `cairomm/win32_font.h`

## 8.35 Cairo::Win32Surface Class Reference

A [Win32Surface](#) provides a way to render within Microsoft Windows.

```
#include <cairomm/win32_surface.h>
```

Inheritance diagram for Cairo::Win32Surface:



## Public Member Functions

- [Win32Surface](#) (`cairo_surface_t *cobject`, `bool has_reference=false`)  
Create a C++ wrapper for the C instance.
- [~Win32Surface](#) () override
- HDC [get\\_dc](#) () const  
Returns the HDC associated with this surface, or NULL if none.
- [RefPtr< ImageSurface > get\\_image](#) ()  
Returns a [ImageSurface](#) that refers to the same bits as the DIB of the Win32 surface.

## Public Member Functions inherited from Cairo::Surface

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & [operator=](#) (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* data, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const



- Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)
  - Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)
  - Writes the [Surface](#) to the write function.*
- [RefPtr< Device >](#) [get\\_device](#) ()
  - This function returns the device for a surface.*
- [cobject \\* cobj](#) ()
  - Provides acces to the underlying C cairo surface.*
- const [cobject \\* cobj](#) () const
  - Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr< Win32Surface >](#) [create](#) (HDC hdc)
  - Creates a cairo surface that targets the given DC.*
- static [RefPtr< Win32Surface >](#) [create\\_with\\_dib](#) ([Format](#) format, int width, int height)
  - Creates a device-independent-bitmap surface not associated with any particular existing surface or device context.*
- static [RefPtr< Win32Surface >](#) [create\\_with\\_ddb](#) (HDC hdc, [Format](#) format, int width, int height)
  - Creates a device-independent-bitmap surface not associated with any particular existing surface or device context.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) &other, [Content](#) content, int width, int height)
  - Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)
  - Create a new surface that is a rectangle within the target surface.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {
  - [IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,
  - [PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,
  - [PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,
  - [XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,
  - [XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,
  - [GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,
  - [QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,
  - [WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,
  - [WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,
  - [BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,
  - [DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,
  - [SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,
  - [OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,
  - [WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,
  - [QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,
  - [SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,
  - [QT](#) = CAIRO\_SURFACE\_TYPE\_QT ,

```

RECORDING = CAIRO_SURFACE_TYPE_RECORDING ,
VG = CAIRO_SURFACE_TYPE_VG ,
GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)

*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*

- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)

*This is the type of function which is called when a backend needs to read data from an input stream.*

- typedef sigc::slot< void()> [SlotDestroy](#)

*For instance, void on\_destroy();.*

- typedef cairo\_surface\_t [cobject](#)

*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)

*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.35.1 Detailed Description

A [Win32Surface](#) provides a way to render within Microsoft Windows.

If you want to draw to the screen within a Microsoft Windows application, you should use this [Surface](#) type.

#### Note

For this [Surface](#) to be available, cairo must have been compiled with Win32 support

### 8.35.2 Constructor & Destructor Documentation

#### 8.35.2.1 Win32Surface()

```

Cairo::Win32Surface::Win32Surface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]

```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a [RefPtr](#).

## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

**8.35.2.2 ~Win32Surface()**

```
Cairo::Win32Surface::~Win32Surface () [override]
```

**8.35.3 Member Function Documentation****8.35.3.1 create()**

```
static RefPtr< Win32Surface > Cairo::Win32Surface::create (
    HDC hdc) [static]
```

Creates a cairo surface that targets the given DC.

The DC will be queried for its initial clip extents, and this will be used as the size of the cairo surface. Also, if the DC is a raster DC, it will be queried for its pixel format and the cairo surface format will be set appropriately.

## Parameters

<i>hdc</i>	the DC to create a surface for.
------------	---------------------------------

## Returns

the newly created surface.

**8.35.3.2 create\_with\_ddb()**

```
static RefPtr< Win32Surface > Cairo::Win32Surface::create_with_ddb (
    HDC hdc,
    Format format,
    int width,
    int height) [static]
```

Creates a device-independent-bitmap surface not associated with any particular existing surface or device context.

The created bitmap will be uninitialized.

## Parameters

<i>hdc</i>	the DC to create a surface for.
<i>format</i>	format of pixels in the surface to create,
<i>width</i>	width of the surface, in pixels.
<i>height</i>	height of the surface, in pixels.

## Returns

the newly created surface.

## Since

1.8

### 8.35.3.3 create\_with\_dib()

```
static RefPtr< Win32Surface > Cairo::Win32Surface::create_with_dib (  
    Format format,  
    int width,  
    int height) [static]
```

Creates a device-independent-bitmap surface not associated with any particular existing surface or device context.  
The created bitmap will be uninitialized.

#### Parameters

<i>format</i>	format of pixels in the surface to create.
<i>width</i>	width of the surface, in pixels.
<i>height</i>	height of the surface, in pixels.

#### Returns

the newly created surface.

#### Since

1.8

### 8.35.3.4 get\_dc()

```
HDC Cairo::Win32Surface::get_dc () const
```

Returns the HDC associated with this surface, or NULL if none.

Also returns NULL if the surface is not a win32 surface.

#### Returns

HDC or NULL if no HDC available.

### 8.35.3.5 get\_image()

```
RefPtr< ImageSurface > Cairo::Win32Surface::get_image ()
```

Returns a [ImageSurface](#) that refers to the same bits as the DIB of the Win32 surface.

If the passed-in win32 surface is not a DIB surface, the returned surface will be NULL

#### Returns

a [ImageSurface](#) (owned by the [Win32Surface](#)), or an empty RefPtr if the win32 surface is not a DIB.

#### Since

1.8

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

- cairomm/win32\_surface.h

## 8.36 Cairo::XlibSurface Class Reference

An [XlibSurface](#) provides a way to render to the X Window System using XLib.

```
#include <cairomm/xlib_surface.h>
```

Inheritance diagram for Cairo::XlibSurface:



### Public Member Functions

- [XlibSurface](#) (cairo\_surface\_t \*[cobject](#), bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [~XlibSurface](#) () override
- void [set\\_size](#) (int width, int height)  
*Informs cairo of the new size of the X Drawable underlying the surface.*
- void [set\\_drawable](#) (Drawable drawable, int width, int height)  
*Informs cairo of a new X Drawable underlying the surface.*
- Drawable [get\\_drawable](#) () const  
*gets the Drawable object associated with this surface*
- const Display \* [get\\_display](#) () const  
*Get the X Display for the underlying X Drawable.*
- Display \* [get\\_display](#) ()  
*Get the X Display for the underlying X Drawable.*
- Screen \* [get\\_screen](#) ()  
*Get the X Screen for the underlying X Drawable.*
- const Screen \* [get\\_screen](#) () const  
*Get the X Screen for the underlying X Drawable.*
- Visual \* [get\\_visual](#) ()  
*Get the X Visual for the underlying X Drawable.*
- const Visual \* [get\\_visual](#) () const  
*Get the X Visual for the underlying X Drawable.*
- int [get\\_depth](#) () const  
*Get the number of bits used to represent each pixel value.*
- int [get\\_height](#) () const  
*Get the height in pixels of the X Drawable underlying the surface.*
- int [get\\_width](#) () const  
*Get the width in pixels of the X Drawable underlying the surface.*
- XRenderPictFormat \* [get\\_xrender\\_format](#) () const  
*Gets the X Render picture format that @surface uses for rendering with the X Render extension.*

## Public Member Functions inherited from Cairo::Surface

- [Surface](#) (cairo\_surface\_t \*cobject, bool has\_reference=false)  
*Create a C++ wrapper for the C instance.*
- [Surface](#) (const [Surface](#) &)=delete
- [Surface](#) & [operator=](#) (const [Surface](#) &)=delete
- virtual [~Surface](#) ()
- const unsigned char \* [get\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned long &length)  
*Return mime data previously attached to surface using the specified mime type.*
- void [set\\_mime\\_data](#) (const **std::string** &mime\_type, unsigned char \* data, unsigned long length, const [SlotDestroy](#) &slot\_destroy)  
*Attach an image in the format mime\_type to surface.*
- void [unset\\_mime\\_data](#) (const **std::string** &mime\_type)  
*Remove the data from a surface.*
- void [get\\_font\\_options](#) ([FontOptions](#) &options) const  
*Retrieves the default font rendering options for the surface.*
- void [finish](#) ()  
*This function finishes the surface and drops all references to external resources.*
- void [flush](#) ()  
*Do any pending drawing for the surface and also restore any temporary modifications cairo has made to the surface's state.*
- void [mark\\_dirty](#) ()  
*Tells cairo to consider the data buffer dirty.*
- void [mark\\_dirty](#) (int x, int y, int width, int height)  
*Marks a rectangular area of the given surface dirty.*
- void [set\\_device\\_offset](#) (double x\_offset, double y\_offset)  
*Sets an offset that is added to the device coordinates determined by the CTM when drawing to surface.*
- void [get\\_device\\_offset](#) (double &x\_offset, double &y\_offset) const  
*Returns a previous device offset set by [set\\_device\\_offset\(\)](#).*
- void [set\\_device\\_scale](#) (double x\_scale, double y\_scale)  
*Sets a scale that is multiplied to the device coordinates determined by the CTM when drawing to surface.*
- void [set\\_device\\_scale](#) (double scale)  
*Sets x and y scale to the same value.*
- void [get\\_device\\_scale](#) (double &x\_scale, double &y\_scale) const  
*Returns a previous device scale set by [set\\_device\\_scale\(\)](#).*
- double [get\\_device\\_scale](#) () const  
*Returns the x and y average of a previous device scale set by [set\\_device\\_scale\(\)](#).*
- void [set\\_fallback\\_resolution](#) (double x\_pixels\_per\_inch, double y\_pixels\_per\_inch)  
*Set the horizontal and vertical resolution for image fallbacks.*
- void [get\\_fallback\\_resolution](#) (double &x\_pixels\_per\_inch, double &y\_pixels\_per\_inch) const  
*This function returns the previous fallback resolution set by [set\\_fallback\\_resolution\(\)](#), or default fallback resolution if never set.*
- [Type](#) [get\\_type](#) () const
- [Content](#) [get\\_content](#) () const  
*This function returns the content type of surface which indicates whether the surface contains color and/or alpha information.*
- void [copy\\_page](#) ()  
*Emits the current page for backends that support multiple pages, but doesn't clear it, so that the contents of the current page will be retained for the next page.*
- void [show\\_page](#) ()  
*Emits and clears the current page for backends that support multiple pages.*
- bool [has\\_show\\_text\\_glyphs](#) () const

- Returns whether the surface supports sophisticated [Context::show\\_text\\_glyphs\(\)](#) operations.*
- void [write\\_to\\_png](#) (const **std::string** &filename)  
*Writes the contents of surface to a new file filename as a PNG image.*
- void [write\\_to\\_png\\_stream](#) (const [SlotWriteFunc](#) &write\_func)  
*Writes the [Surface](#) to the write function.*
- [RefPtr< Device >](#) [get\\_device](#) ()  
*This function returns the device for a surface.*
- [cobject \\* cobj](#) ()  
*Provides acces to the underlying C cairo surface.*
- const [cobject \\* cobj](#) () const  
*Provides acces to the underlying C cairo surface.*

### Static Public Member Functions

- static [RefPtr< XlibSurface >](#) [create](#) (Display \*dpy, Drawable drawable, Visual \*visual, int width, int height)  
*Creates an Xlib surface that draws to the given drawable.*
- static [RefPtr< XlibSurface >](#) [create](#) (Display \*dpy, Pixmap bitmap, Screen \*screen, int width, int height)  
*Creates an Xlib surface that draws to the given bitmap.*
- static [Cairo::RefPtr< Cairo::XlibSurface >](#) [create\\_with\\_xrender\\_format](#) (Display \*dpy, Drawable drawable, Screen \*screen, XRenderPictFormat \*format, int width, int height)  
*Creates an Xlib surface that draws to the given drawable.*

### Static Public Member Functions inherited from [Cairo::Surface](#)

- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) other, [Content](#) content, int width, int height)  
*Create a new surface that is as compatible as possible with an existing surface.*
- static [RefPtr< Surface >](#) [create](#) (const [RefPtr< Surface >](#) &target, double x, double y, double width, double height)  
*Create a new surface that is a rectangle within the target surface.*

### Additional Inherited Members

### Public Types inherited from [Cairo::Surface](#)

- enum class [Type](#) {  
[IMAGE](#) = CAIRO\_SURFACE\_TYPE\_IMAGE ,  
[PDF](#) = CAIRO\_SURFACE\_TYPE\_PDF ,  
[PS](#) = CAIRO\_SURFACE\_TYPE\_PS ,  
[XLIB](#) = CAIRO\_SURFACE\_TYPE\_XLIB ,  
[XCB](#) = CAIRO\_SURFACE\_TYPE\_XCB ,  
[GLITZ](#) = CAIRO\_SURFACE\_TYPE\_GLITZ ,  
[QUARTZ](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ ,  
[WIN32](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[WIN32\\_SURFACE](#) = CAIRO\_SURFACE\_TYPE\_WIN32 ,  
[BEOS](#) = CAIRO\_SURFACE\_TYPE\_BEOS ,  
[DIRECTFB](#) = CAIRO\_SURFACE\_TYPE\_DIRECTFB ,  
[SVG](#) = CAIRO\_SURFACE\_TYPE\_SVG ,  
[OS2](#) = CAIRO\_SURFACE\_TYPE\_OS2 ,  
[WIN32\\_PRINTING](#) = CAIRO\_SURFACE\_TYPE\_WIN32\_PRINTING ,  
[QUARTZ\\_IMAGE](#) = CAIRO\_SURFACE\_TYPE\_QUARTZ\_IMAGE ,  
[SCRIPT](#) = CAIRO\_SURFACE\_TYPE\_SCRIPT ,

```

QT = CAIRO_SURFACE_TYPE_QT ,
RECORDING = CAIRO_SURFACE_TYPE_RECORDING ,
VG = CAIRO_SURFACE_TYPE_VG ,
GL = CAIRO_SURFACE_TYPE_GL ,
DRM = CAIRO_SURFACE_TYPE_DRM ,
TEE = CAIRO_SURFACE_TYPE_TEE ,
XML = CAIRO_SURFACE_TYPE_XML ,
SKIA = CAIRO_SURFACE_TYPE_SKIA ,
SUBSURFACE = CAIRO_SURFACE_TYPE_SUBSURFACE }

```

*Cairo::Surface::Type is used to describe the type of a given surface.*

- enum class [Format](#) {  
[ARGB32](#) = CAIRO\_FORMAT\_ARGB32 ,  
[RGB24](#) = CAIRO\_FORMAT\_RGB24 ,  
[A8](#) = CAIRO\_FORMAT\_A8 ,  
[A1](#) = CAIRO\_FORMAT\_A1 ,  
[RGB16\\_565](#) = CAIRO\_FORMAT\_RGB16\_565 }

*Format is used to identify the memory format of image data.*

- typedef sigc::slot< [ErrorStatus](#)(const unsigned char \*, unsigned int)> [SlotWriteFunc](#)

*For example: `ErrorStatus my_write_func(unsigned char* data, unsigned int length);`*

- typedef sigc::slot< [ErrorStatus](#)(unsigned char \*, unsigned int)> [SlotReadFunc](#)

*This is the type of function which is called when a backend needs to read data from an input stream.*

- typedef sigc::slot< void()> [SlotDestroy](#)

*For instance, void on\_destroy();.*

- typedef cairo\_surface\_t [cobject](#)

*The underlying C cairo surface type.*

## Protected Attributes inherited from [Cairo::Surface](#)

- [cobject](#) \* [m\\_cobject](#)

*The underlying C cairo surface type that is wrapped by this [Surface](#).*

### 8.36.1 Detailed Description

An [XlibSurface](#) provides a way to render to the X Window System using XLib.

If you want to draw to the screen within an application that uses the X Window system, you should use this [Surface](#) type.

#### Note

For this surface to be available, cairo must have been compiled with support for XLib Surfaces

### 8.36.2 Constructor & Destructor Documentation

#### 8.36.2.1 [XlibSurface](#)()

```

Cairo::XlibSurface::XlibSurface (
    cairo_surface_t * cobject,
    bool has_reference = false) [explicit]

```

Create a C++ wrapper for the C instance.

This C++ instance should then be given to a [RefPtr](#).



## Parameters

<i>cobject</i>	The C instance.
<i>has_reference</i>	whether we already have a reference. Otherwise, the constructor will take an extra reference.

## 8.36.2.2 ~XlibSurface()

```
Cairo::XlibSurface::~XlibSurface () [override]
```

## 8.36.3 Member Function Documentation

## 8.36.3.1 create() [1/2]

```
static RefPtr< XlibSurface > Cairo::XlibSurface::create (
    Display * dpy,
    Drawable drawable,
    Visual * visual,
    int width,
    int height) [static]
```

Creates an Xlib surface that draws to the given drawable.

The way that colors are represented in the drawable is specified by the provided visual.

## Note

If drawable is a Window, then the function `cairo_xlib_surface_set_size` must be called whenever the size of the window changes.

## Parameters

<i>dpy</i>	an X Display
<i>drawable</i>	an X Drawable, (a Pixmap or a Window)
<i>visual</i>	the visual to use for drawing to drawable. The depth of the visual must match the depth of the drawable. Currently, only TrueColor visuals are fully supported.
<i>width</i>	the current width of drawable.
<i>height</i>	the current height of drawable.

## Returns

A RefPtr to the newly created surface

## 8.36.3.2 create() [2/2]

```
static RefPtr< XlibSurface > Cairo::XlibSurface::create (
    Display * dpy,
    Pixmap bitmap,
    Screen * screen,
    int width,
    int height) [static]
```

Creates an Xlib surface that draws to the given bitmap.

This will be drawn to as a `CAIRO_FORMAT_A1` object.

## Parameters

<i>dpy</i>	an X Display
<i>bitmap</i>	an X Drawable, (a depth-1 Pixmap)
<i>screen</i>	the X Screen associated with bitmap
<i>width</i>	the current width of bitmap.
<i>height</i>	the current height of bitmap.

## Returns

A RefPtr to the newly created surface

**8.36.3.3 create\_with\_xrender\_format()**

```
static Cairo::RefPtr< Cairo::XlibSurface > Cairo::XlibSurface::create_with_xrender_format (
    Display * dpy,
    Drawable drawable,
    Screen * screen,
    XRenderPictFormat * format,
    int width,
    int height) [static]
```

Creates an Xlib surface that draws to the given drawable.

The way that colors are represented in the drawable is specified by the provided picture format.

Note: If @drawable is a Window, then the function [set\\_size\(\)](#) must be called whenever the size of the window changes.

## Parameters

<i>dpy</i>	an X Display
<i>drawable</i>	an X Drawable, (a Pixmap or a Window)
<i>screen</i>	the X Screen associated with @drawable
<i>format</i>	the picture format to use for drawing to @drawable. The depth of @format must match the depth of the drawable.
<i>width</i>	the current width of @drawable.
<i>height</i>	the current height of @drawable.

## Returns

the newly created surface

**8.36.3.4 get\_depth()**

```
int Cairo::XlibSurface::get_depth () const
```

Get the number of bits used to represent each pixel value.

**8.36.3.5 get\_display() [1/2]**

```
Display * Cairo::XlibSurface::get_display ()
```

Get the X Display for the underlying X Drawable.

**8.36.3.6 get\_display() [2/2]**

```
const Display * Cairo::XlibSurface::get_display () const
```

Get the X Display for the underlying X Drawable.

**8.36.3.7 get\_drawable()**

```
Drawable Cairo::XlibSurface::get_drawable () const
```

gets the Drawable object associated with this surface

**8.36.3.8 get\_height()**

```
int Cairo::XlibSurface::get_height () const
```

Get the height in pixels of the X Drawable underlying the surface.

**8.36.3.9 get\_screen() [1/2]**

```
Screen * Cairo::XlibSurface::get_screen ()
```

Get the X Screen for the underlying X Drawable.

**8.36.3.10 get\_screen() [2/2]**

```
const Screen * Cairo::XlibSurface::get_screen () const
```

Get the X Screen for the underlying X Drawable.

**8.36.3.11 get\_visual() [1/2]**

```
Visual * Cairo::XlibSurface::get_visual ()
```

Get the X Visual for the underlying X Drawable.

**8.36.3.12 get\_visual() [2/2]**

```
const Visual * Cairo::XlibSurface::get_visual () const
```

Get the X Visual for the underlying X Drawable.

**8.36.3.13 get\_width()**

```
int Cairo::XlibSurface::get_width () const
```

Get the width in pixels of the X Drawable underlying the surface.

**8.36.3.14 get\_xrender\_format()**

```
XRenderPictFormat * Cairo::XlibSurface::get_xrender_format () const
```

Gets the X Render picture format that @surface uses for rendering with the X Render extension.

If the surface was created by `cairo_xlib_surface_create_with_xrender_format()` originally, the return value is the format passed to that constructor.

Return value: the `XRenderPictFormat*` associated with @surface, or `NULL` if the surface is not an xlib surface or if the X Render extension is not available.

Since: 1.6

**8.36.3.15 set\_drawable()**

```
void Cairo::XlibSurface::set_drawable (
    Drawable drawable,
    int width,
    int height)
```

Informs cairo of a new X Drawable underlying the surface.

The drawable must match the display, screen and format of the existing drawable or the application will get X protocol errors and will probably terminate. No checks are done by this function to ensure this compatibility.

**Parameters**

<i>drawable</i>	the new drawable for the surface
<i>width</i>	the width of the new drawable
<i>height</i>	the height of the new drawable

**8.36.3.16 set\_size()**

```
void Cairo::XlibSurface::set_size (
    int width,
    int height)
```

Informs cairo of the new size of the X Drawable underlying the surface.

For a surface created for a Window (rather than a Pixmap), this function must be called each time the size of the window changes. (For a subwindow, you are normally resizing the window yourself, but for a toplevel window, it is necessary to listen for `ConfigureNotify` events.)

A Pixmap can never change size, so it is never necessary to call this function on a surface created for a Pixmap.

## Parameters

<i>width</i>	the new width of the surface
<i>height</i>	the new height of the surface

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

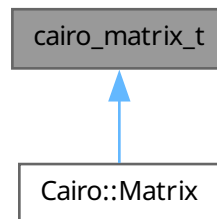
- `cairomm/xlib_surface.h`

## 8.37 cairo\_matrix\_t Class Reference

See the [cairo\\_matrix\\_t reference](#) in the cairo manual for more information.

```
#include <cairomm/matrix.h>
```

Inheritance diagram for `cairo_matrix_t`:



### 8.37.1 Detailed Description

See the [cairo\\_matrix\\_t reference](#) in the cairo manual for more information.

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

- `cairomm/matrix.h`

## 8.38 hash\_load\_check\_resize\_trigger\_size\_base Class Reference

Inheritance diagram for `hash_load_check_resize_trigger_size_base`:



The documentation for this class was generated from the following files:

## 8.39 lu\_counter\_policy\_base Class Reference

Inheritance diagram for lu\_counter\_policy\_base:



The documentation for this class was generated from the following files:

## 8.40 mask\_based\_range\_hashing Class Reference

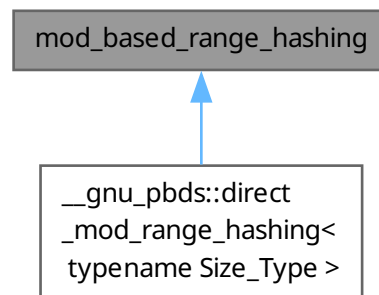
Inheritance diagram for mask\_based\_range\_hashing:



The documentation for this class was generated from the following files:

## 8.41 mod\_based\_range\_hashing Class Reference

Inheritance diagram for mod\_based\_range\_hashing:



The documentation for this class was generated from the following files:





## Chapter 9

# Examples

### 9.1 toy-text.cc

A relatively simple example of using [Cairo::ToyFontFace](#).

A relatively simple example of using [Cairo::ToyFontFace](#)

```
#include <cairomm/cairomm.h>

const double HEIGHT = 200.0;
const double WIDTH = 400.0;
const double FONT_SIZE = 64.0;
const double TEXT_ORIGIN_Y = (HEIGHT / 2.0) + (FONT_SIZE / 2.0);
const double TEXT_ORIGIN_X = 50.0; // arbitrary

int main(int, char**)
{
    auto surface =
        Cairo::ImageSurface::create(Cairo::Surface::Format::ARGB32, WIDTH, HEIGHT);
    auto cr = Cairo::Context::create(surface);
    // fill background in white
    cr->set_source_rgb(1.0, 1.0, 1.0);
    cr->paint();

    // draw a little dot at the point where text will be drawn
    cr->arc(TEXT_ORIGIN_X, TEXT_ORIGIN_Y, FONT_SIZE / 4.0, 0, 2*M_PI);
    cr->set_source_rgba(0.0, 1.0, 0.0, 0.5);
    cr->fill();

    // draw the text
    cr->move_to(TEXT_ORIGIN_X, TEXT_ORIGIN_Y);
    cr->set_source_rgb(0.8, 0.2, 0.2);
    auto font =
        Cairo::ToyFontFace::create("Bitstream Charter",
                                   Cairo::ToyFontFace::Slant::ITALIC,
                                   Cairo::ToyFontFace::Weight::BOLD);

    cr->set_font_face(font);
    cr->set_font_size(FONT_SIZE);
    cr->show_text("cairomm!");
    surface->write_to_png("toy-text.png");
    return 0;
}
```

### 9.2 user-font.cc

A relatively simple example of using [Cairo::UserFontFace](#).

A relatively simple example of using [Cairo::UserFontFace](#)

```
#include <cairomm/cairomm.h>
#include <iostream>
```

```

#include <map>

const double HEIGHT = 200.0;
const double WIDTH = 400.0;
const double FONT_SIZE = 64.0;
const double TEXT_ORIGIN_Y = (HEIGHT / 2.0) + (FONT_SIZE / 2.0);
const double TEXT_ORIGIN_X = 50.0; // arbitrary
const double GLYPH_SPACING = 0.1;

struct GlyphBounds
{
    unsigned long glyph;
    double width;
    double height;
};

// an array that stores the bounds of the glyphs that we're going to draw
static const GlyphBounds glyphs[] =
{
    { 'c', 0.45, 0.5 },
    { 'a', 0.45, 0.5 },
    { 'i', 0.2, 0.75 },
    { 'r', 0.4, 0.5 },
    { 'o', 0.44, 0.5 },
    { 'm', 0.75, 0.5 },
    { '!', 0.2, 0.75 }
};

// A *very* simple font that just draws a box for every glyph
class BoxFontFace : public Cairo::UserFontFace
{
public:
    // Derived user font classes should have a factory method to create an object
    // and return it with a RefPtr
    static Cairo::RefPtr<BoxFontFace> create()
    {
        return Cairo::make_refptr_for_instance<BoxFontFace>(new BoxFontFace());
    }

    Cairo::ErrorStatus
    init(const Cairo::RefPtr<Cairo::ScaledFont>& /*scaled_font*/,
         const Cairo::RefPtr<Cairo::Context>& /*cr*/,
         Cairo::FontExtents &extents) override
    {
        double max = 0;
        for (unsigned int i = 0; i < sizeof (glyphs) / sizeof (GlyphBounds); ++i) {
            if (glyphs[i].width > max)
                max = glyphs[i].width;
        }
        // add some spacing between characters
        max += GLYPH_SPACING;
        extents.max_x_advance = max;
        return CAIRO_STATUS_SUCCESS;
    }

    Cairo::ErrorStatus
    unicode_to_glyph (const Cairo::RefPtr<Cairo::ScaledFont>& /*scaled_font*/,
                     unsigned long unicode, unsigned long& glyph) override
    {
        glyph = 0;
        // yes this is a stupid and inefficient way to do this but we only have a few
        // glyphs and this is just demonstration code
        for (unsigned int i = 0; i < sizeof (glyphs) / sizeof (GlyphBounds); ++i) {
            if (glyphs[i].glyph == unicode) {
                // glyph 0 is often a special glyph-not-found value, so offset it by 1
                glyph = i+1;
                break;
            }
        }
        return CAIRO_STATUS_SUCCESS;
    }

    Cairo::ErrorStatus
    render_glyph(const Cairo::RefPtr<Cairo::ScaledFont>& /*scaled_font*/,
                 unsigned long glyph,
                 const Cairo::RefPtr<Cairo::Context>& cr,
                 Cairo::TextExtents& metrics) override
    {
        // check that the glyph is in our table
        if (glyph >= 1 && glyph <= sizeof(glyphs)/sizeof(GlyphBounds)) {
            cr->set_line_width(0.05);
            // Need a negative Y value since the text origin is at the bottom left point
            // and cairo's positive Y axis is down and we want to draw up
            cr->rectangle(0.0, 0.0, glyphs[glyph-1].width, -glyphs[glyph-1].height);
            cr->stroke();
            metrics.x_advance = glyphs[glyph-1].width + GLYPH_SPACING;
        }
    }
}

```

```

    return CAIRO_STATUS_SUCCESS;
}

protected:
    // FontFace is a ref-counted object, so the constructor should be protected so
    // it is not created without a refptr to manage it. See the create() method
    BoxFontFace() : UserFontFace() { }
};

int main(int, char**)
{
    auto surface =
        Cairo::ImageSurface::create(Cairo::Surface::Format::ARGB32, WIDTH, HEIGHT);
    auto cr = Cairo::Context::create(surface);
    // fill background in white
    cr->set_source_rgb(1.0, 1.0, 1.0);
    cr->paint();

    // draw a little dot at the point where text will be drawn
    cr->arc(TEXT_ORIGIN_X, TEXT_ORIGIN_Y, FONT_SIZE / 4.0, 0, 2*M_PI);
    cr->set_source_rgba(0.0, 1.0, 0.0, 0.5);
    cr->fill();

    // draw the text
    cr->move_to(TEXT_ORIGIN_X, TEXT_ORIGIN_Y);
    cr->set_source_rgb(0.8, 0.2, 0.2);
    auto font = BoxFontFace::create();
    cr->set_font_face(font);
    cr->set_font_size(FONT_SIZE);
    cr->show_text("cairomm!");

    // Now show it with the toy text API to demonstrate how the glyphs match up
    cr->move_to(TEXT_ORIGIN_X, TEXT_ORIGIN_Y);
    cr->set_source_rgba(0.2, 0.2, 0.2, 0.3);
    auto toy_font =
        Cairo::ToyFontFace::create("Bitstream Charter",
                                    Cairo::ToyFontFace::Slant::NORMAL,
                                    Cairo::ToyFontFace::Weight::BOLD);

    cr->set_font_face(toy_font);
    cr->set_font_size(FONT_SIZE);
    cr->show_text("cairomm!");

    const char* filename = "user-font.png";
    try {
        surface->write_to_png(filename);
        std::cout << "Wrote Image " << filename << std::endl;
        return 0;
    } catch (const std::exception& e)
    {
        std::cout << "*** Unable to write Image " << filename << std::endl;
        return 1;
    }
}

```

## 9.3 image-surface.cc

An example of using [Cairo::ImageSurface](#) class to render to PNG.

An example of using [Cairo::ImageSurface](#) class to render to PNG

```

/* M_PI is defined in math.h in the case of Microsoft Visual C++, Solaris,
 * et. al.
 */

#include <string>
#include <iostream>
#include <cairommconfig.h>
#include <cairomm/context.h>
#include <cairomm/surface.h>

#include <cmath>

int main()
{
    auto surface =
        Cairo::ImageSurface::create(Cairo::Surface::Format::ARGB32, 600, 400);

    auto cr = Cairo::Context::create(surface);

    cr->save(); // save the state of the context
    cr->set_source_rgb(0.86, 0.85, 0.47);

```

```

cr->paint();    // fill image with the color
cr->restore();  // color is back to black now

cr->save();
// draw a border around the image
cr->set_line_width(20.0);    // make the line wider
cr->rectangle(0.0, 0.0, surface->get_width(), surface->get_height());
cr->stroke();

cr->set_source_rgba(0.0, 0.0, 0.0, 0.7);
// draw a circle in the center of the image
cr->arc(surface->get_width() / 2.0, surface->get_height() / 2.0,
        surface->get_height() / 4.0, 0.0, 2.0 * M_PI);
cr->stroke();

// draw a diagonal line
cr->move_to(surface->get_width() / 4.0, surface->get_height() / 4.0);
cr->line_to(surface->get_width() * 3.0 / 4.0, surface->get_height() * 3.0 / 4.0);
cr->stroke();
cr->restore();

#ifdef CAIRO_HAS_PNG_FUNCTIONS

    std::string filename = "image.png";
    surface->write_to_png(filename);

    std::cout << "Wrote png file \"" << filename << "\"" << std::endl;

#else

    std::cout << "You must compile cairo with PNG support for this example to work."
              << std::endl;

#endif
}

```

## 9.4 pdf-surface.cc

An example of using [Cairo::PdfSurface](#) class to render to PDF.

An example of using [Cairo::PdfSurface](#) class to render to PDF

```

/* M_PI is defined in math.h in the case of Microsoft Visual C++, Solaris,
 * et. al.
 */

#include <string>
#include <iostream>
#include <cairommconfig.h>
#include <cairomm/context.h>
#include <cairomm/surface.h>

#include <cmath>

int main()
{
#ifdef CAIRO_HAS_PDF_SURFACE

    std::string filename = "image.pdf";
    int width = 600;
    int height = 400;
    auto surface =
        Cairo::PdfSurface::create(filename, width, height);

    auto cr = Cairo::Context::create(surface);

    cr->save(); // save the state of the context
    cr->set_source_rgb(0.86, 0.85, 0.47);
    cr->paint(); // fill image with the color
    cr->restore(); // color is back to black now

    cr->save();
    // draw a border around the image
    cr->set_line_width(20.0);    // make the line wider
    cr->rectangle(0.0, 0.0, cairo_image_surface_get_width(surface->cobj()), height);
    cr->stroke();

    cr->set_source_rgba(0.0, 0.0, 0.0, 0.7);
    // draw a circle in the center of the image
    cr->arc(width / 2.0, height / 2.0,
            height / 4.0, 0.0, 2.0 * M_PI);

```

```

    cr->stroke();

    // draw a diagonal line
    cr->move_to(width / 4.0, height / 4.0);
    cr->line_to(width * 3.0 / 4.0, height * 3.0 / 4.0);
    cr->stroke();
    cr->restore();

    cr->show_page();

    std::cout << "Wrote PDF file \"" << filename << "\"" << std::endl;
    return 0;

#else

    std::cout << "You must compile cairo with PDF support for this example to work."
              << std::endl;
    return 1;

#endif
}

```

## 9.5 ps-surface.cc

An example of using [Cairo::PsSurface](#) class to render to PostScript.

An example of using [Cairo::PsSurface](#) class to render to PostScript

```

/* M_PI is defined in math.h in the case of Microsoft Visual C++, Solaris,
 * et. al.
 */

#include <string>
#include <iostream>
#include <cairommconfig.h>
#include <cairomm/context.h>
#include <cairomm/surface.h>

#include <cmath>

int main()
{
#ifdef CAIRO_HAS_PS_SURFACE

    std::string filename = "image.ps";
    double width = 600;
    double height = 400;
    auto surface =
        Cairo::PsSurface::create(filename, width, height);

    auto cr = Cairo::Context::create(surface);

    cr->save(); // save the state of the context
    cr->set_source_rgb(0.86, 0.85, 0.47);
    cr->paint(); // fill image with the color
    cr->restore(); // color is back to black now

    cr->save();
    // draw a border around the image
    cr->set_line_width(20.0); // make the line wider
    cr->rectangle(0.0, 0.0, cairo_image_surface_get_width(surface->cobj()), height);
    cr->stroke();

    cr->set_source_rgba(0.0, 0.0, 0.0, 0.7);
    // draw a circle in the center of the image
    cr->arc(width / 2.0, height / 2.0,
           height / 4.0, 0.0, 2.0 * M_PI);
    cr->stroke();

    // draw a diagonal line
    cr->move_to(width / 4.0, height / 4.0);
    cr->line_to(width * 3.0 / 4.0, height * 3.0 / 4.0);
    cr->stroke();
    cr->restore();

    cr->show_page();

    std::cout << "Wrote PostScript file \"" << filename << "\"" << std::endl;
    return 0;

#else

    std::cout << "You must compile cairo with PS (Postscript) support for this example to work."

```

```

        « std::endl;
    return 1;
#endif
}

```

## 9.6 svg-surface.cc

An example of using [Cairo::SvgSurface](#) class to render to SVG.

An example of using [Cairo::SvgSurface](#) class to render to SVG

```

/* M_PI is defined in math.h in the case of Microsoft Visual C++, Solaris,
 * et. al.
 */

#include <string>
#include <iostream>
#include <cairommconfig.h>
#include <cairomm/context.h>
#include <cairomm/surface.h>

#include <cmath>

int main()
{
#ifdef CAIRO_HAS_SVG_SURFACE

    std::string filename = "image.svg";
    double width = 600;
    double height = 400;
    auto surface =
        Cairo::SvgSurface::create(filename, width, height);

    auto cr = Cairo::Context::create(surface);

    cr->save(); // save the state of the context
    cr->set_source_rgb(0.86, 0.85, 0.47);
    cr->paint(); // fill image with the color
    cr->restore(); // color is back to black now

    cr->save();
    // draw a border around the image
    cr->set_line_width(20.0); // make the line wider
    cr->rectangle(0.0, 0.0, cairo_image_surface_get_width(surface->cobj()), height);
    cr->stroke();

    cr->set_source_rgba(0.0, 0.0, 0.0, 0.7);
    // draw a circle in the center of the image
    cr->arc(width / 2.0, height / 2.0,
           height / 4.0, 0.0, 2.0 * M_PI);
    cr->stroke();

    // draw a diagonal line
    cr->move_to(width / 4.0, height / 4.0);
    cr->line_to(width * 3.0 / 4.0, height * 3.0 / 4.0);
    cr->stroke();
    cr->restore();

    cr->show_page();

    std::cout << "Wrote SVG file \"" << filename << "\"" << std::endl;
    return 0;
#else

    std::cout << "You must compile cairo with SVG support for this example to work."
        << std::endl;
    return 1;
#endif
}

```

# Index

- ~Context
  - Cairo::Context, [30](#)
- ~Device
  - Cairo::Device, [69](#)
- ~FontFace
  - Cairo::FontFace, [74](#)
- ~FontOptions
  - Cairo::FontOptions, [78](#)
- ~GlitzSurface
  - Cairo::GlitzSurface, [91](#)
- ~Gradient
  - Cairo::Gradient, [94](#)
- ~ImageSurface
  - Cairo::ImageSurface, [100](#)
- ~LinearGradient
  - Cairo::LinearGradient, [106](#)
- ~Lock
  - Cairo::Device::Lock, [72](#)
- ~Path
  - Cairo::Path, [116](#)
- ~Pattern
  - Cairo::Pattern, [119](#)
- ~PdfSurface
  - Cairo::PdfSurface, [126](#)
- ~PsSurface
  - Cairo::PsSurface, [133](#)
- ~QuartzSurface
  - Cairo::QuartzSurface, [143](#)
- ~RadialGradient
  - Cairo::RadialGradient, [147](#)
- ~RecordingSurface
  - Cairo::RecordingSurface, [152](#)
- ~Region
  - Cairo::Region, [156](#)
- ~SaveGuard
  - Cairo::SaveGuard, [160](#)
- ~ScaledFont
  - Cairo::ScaledFont, [163](#)
- ~SolidPattern
  - Cairo::SolidPattern, [169](#)
- ~Surface
  - Cairo::Surface, [178](#)
- ~SurfacePattern
  - Cairo::SurfacePattern, [191](#)
- ~SvgSurface
  - Cairo::SvgSurface, [195](#)
- ~UserFontFace
  - Cairo::UserFontFace, [203](#)
- ~Win32PrintingSurface
  - Cairo::Win32PrintingSurface, [214](#)
- ~Win32Surface
  - Cairo::Win32Surface, [223](#)
- ~XlibSurface
  - Cairo::XlibSurface, [229](#)
- ~logic\_error
  - Cairo::logic\_error, [108](#)
- A1
  - Cairo::Surface, [175](#)
- A8
  - Cairo::Surface, [175](#)
- acquire
  - Cairo::Device, [70](#)
- ADD
  - Cairo::Context, [30](#)
- add\_color\_stop\_rgb
  - Cairo::Gradient, [94](#)
- add\_color\_stop\_rgba
  - Cairo::Gradient, [94](#)
- alpha
  - Cairo::ColorStop, [21](#)
- Antialias
  - Cairo, [17](#)
- ANTIALIAS\_DEFAULT
  - Cairo, [17](#)
- ANTIALIAS\_GRAY
  - Cairo, [17](#)
- ANTIALIAS\_NONE
  - Cairo, [17](#)
- ANTIALIAS\_SUBPIXEL
  - Cairo, [17](#)
- append\_path
  - Cairo::Context, [31](#)
- arc
  - Cairo::Context, [31](#)
- arc\_negative
  - Cairo::Context, [32](#)
- ARGB32
  - Cairo::Surface, [175](#)
- ATOP
  - Cairo::Context, [30](#)
- begin\_new\_path
  - Cairo::Context, [32](#)
- begin\_new\_sub\_path
  - Cairo::Context, [32](#)
- BEOS
  - Cairo::Surface, [176](#)
- BEST

- Cairo::SurfacePattern, 190
- BEVEL
  - Cairo::Context, 29
- BILINEAR
  - Cairo::SurfacePattern, 190
- blue
  - Cairo::ColorStop, 21
- BOLD
  - Cairo::ToyFontFace, 200
- BUTT
  - Cairo::Context, 29
- Cairo, 13
  - Antialias, 17
  - ANTIALIAS\_DEFAULT, 17
  - ANTIALIAS\_GRAY, 17
  - ANTIALIAS\_NONE, 17
  - ANTIALIAS\_SUBPIXEL, 17
  - Content, 17
  - CONTENT\_ALPHA, 17
  - CONTENT\_COLOR, 17
  - CONTENT\_COLOR\_ALPHA, 17
  - FONT\_TYPE\_FT, 18
  - FONT\_TYPE\_QUARTZ, 18
  - FONT\_TYPE\_TOY, 18
  - FONT\_TYPE\_USER, 18
  - FONT\_TYPE\_WIN32, 18
  - FontExtents, 16
  - FontType, 17
  - FT\_SYNTHESIZE\_BOLD, 18
  - FT\_SYNTHESIZE\_OBLIQUE, 18
  - FtSynthesize, 18
  - Glyph, 16
  - make\_refptr\_for\_instance, 20
  - operator&, 20
  - operator|, 20
  - PDF\_VERSION\_1\_4, 18
  - PDF\_VERSION\_1\_5, 18
  - PdfVersion, 18
  - PS\_LEVEL\_2, 19
  - PS\_LEVEL\_3, 19
  - PsLevel, 18
  - Rectangle, 16
  - RectangleInt, 16
  - RefPtr, 16
  - SUBPIXEL\_ORDER\_BGR, 19
  - SUBPIXEL\_ORDER\_DEFAULT, 19
  - SUBPIXEL\_ORDER\_RGB, 19
  - SUBPIXEL\_ORDER\_VBGR, 19
  - SUBPIXEL\_ORDER\_VRGB, 19
  - SubpixelOrder, 19
  - SVG\_VERSION\_1\_1, 19
  - SVG\_VERSION\_1\_2, 19
  - SvgVersion, 19
  - TEXT\_CLUSTER\_FLAG\_BACKWARD, 19
  - TextCluster, 16
  - TextClusterFlags, 19
  - TextExtents, 17
- Cairo::ColorStop, 21
  - alpha, 21
  - blue, 21
  - green, 21
  - offset, 21
  - red, 21
- Cairo::Context, 22
  - ~Context, 30
  - ADD, 30
  - append\_path, 31
  - arc, 31
  - arc\_negative, 32
  - ATOP, 30
  - begin\_new\_path, 32
  - begin\_new\_sub\_path, 32
  - BEVEL, 29
  - BUTT, 29
  - CLEAR, 30
  - clip, 32
  - clip\_preserve, 33
  - close\_path, 33
  - cobj, 33, 34
  - cobject, 28
  - Context, 30
  - copy\_clip\_rectangle\_list, 34
  - copy\_page, 34
  - copy\_path, 34
  - copy\_path\_flat, 35
  - create, 35
  - curve\_to, 35
  - DEST, 30
  - DEST\_ATOP, 30
  - DEST\_IN, 30
  - DEST\_OUT, 30
  - DEST\_OVER, 30
  - device\_to\_user, 36
  - device\_to\_user\_distance, 36
  - EVEN\_ODD, 29
  - fill, 36
  - fill\_preserve, 36
  - FillRule, 28
  - get\_antialias, 37
  - get\_clip\_extents, 37
  - get\_current\_point, 37
  - get\_dash, 38
  - get\_fill\_extents, 38
  - get\_fill\_rule, 38
  - get\_font\_extents, 39
  - get\_font\_face, 39
  - get\_font\_matrix, 39
  - get\_font\_options, 39
  - get\_glyph\_extents, 40
  - get\_group\_target, 40
  - get\_line\_cap, 41
  - get\_line\_join, 41
  - get\_line\_width, 41
  - get\_matrix, 41
  - get\_miter\_limit, 42
  - get\_operator, 42



- get\_path\_extents, [42](#)
- get\_scaled\_font, [42](#)
- get\_source, [43](#)
- get\_source\_for\_surface, [43](#)
- get\_stroke\_extents, [43](#)
- get\_target, [44](#)
- get\_text\_extents, [44](#)
- get\_tolerance, [45](#)
- glyph\_path, [45](#)
- has\_current\_point, [45](#)
- IN, [30](#)
- in\_clip, [45](#)
- in\_fill, [46](#)
- in\_stroke, [46](#)
- line\_to, [47](#)
- LineCap, [29](#)
- LineJoin, [29](#)
- m\_cobject, [67](#)
- mask, [47](#)
- MITER, [29](#)
- move\_to, [48](#)
- Operator, [29](#)
- operator=, [48](#)
- OUT, [30](#)
- OVER, [30](#)
- paint, [48](#)
- paint\_with\_alpha, [48](#)
- pop\_group, [48](#)
- pop\_group\_to\_source, [49](#)
- push\_group, [49](#)
- push\_group\_with\_content, [50](#)
- rectangle, [50](#)
- rel\_curve\_to, [50](#)
- rel\_line\_to, [51](#)
- rel\_move\_to, [51](#)
- reset\_clip, [52](#)
- restore, [52](#)
- rotate, [52](#)
- rotate\_degrees, [53](#)
- ROUND, [29](#)
- SATURATE, [30](#)
- save, [53](#)
- scale, [53](#)
- select\_font\_face, [53](#)
- set\_antialias, [54](#)
- set\_dash, [55](#)
- set\_fill\_rule, [55](#)
- set\_font\_face, [56](#)
- set\_font\_matrix, [56](#)
- set\_font\_options, [56](#)
- set\_font\_size, [56](#)
- set\_identity\_matrix, [58](#)
- set\_line\_cap, [58](#)
- set\_line\_join, [58](#)
- set\_line\_width, [58](#)
- set\_matrix, [59](#)
- set\_miter\_limit, [59](#)
- set\_operator, [60](#)
- set\_scaled\_font, [60](#)
- set\_source, [60](#), [61](#)
- set\_source\_rgb, [61](#)
- set\_source\_rgba, [62](#)
- set\_tolerance, [62](#)
- show\_glyphs, [63](#)
- show\_page, [63](#)
- show\_text, [63](#)
- show\_text\_glyphs, [63](#)
- SOURCE, [30](#)
- SQUARE, [29](#)
- stroke, [64](#)
- stroke\_preserve, [65](#)
- text\_path, [65](#)
- transform, [66](#)
- translate, [66](#)
- unset\_dash, [66](#)
- user\_to\_device, [66](#)
- user\_to\_device\_distance, [67](#)
- WINDING, [29](#)
- XOR, [30](#)
- Cairo::Device, [67](#)
  - ~Device, [69](#)
  - acquire, [70](#)
  - cobj, [70](#)
  - cobject, [69](#)
  - Device, [69](#)
  - DeviceType, [69](#)
  - DRM, [69](#)
  - finish, [70](#)
  - flush, [70](#)
  - get\_type, [71](#)
  - GL, [69](#)
  - m\_cobject, [71](#)
  - reference, [71](#)
  - release, [71](#)
  - SCRIPT, [69](#)
  - unreference, [71](#)
  - XCB, [69](#)
  - XLIB, [69](#)
  - XML, [69](#)
- Cairo::Device::Lock, [71](#)
  - ~Lock, [72](#)
  - Lock, [72](#)
- Cairo::FontFace, [73](#)
  - ~FontFace, [74](#)
  - cobj, [74](#)
  - cobject, [74](#)
  - FontFace, [74](#)
  - get\_type, [75](#)
  - m\_cobject, [75](#)
  - operator=, [75](#)
  - reference, [75](#)
  - unreference, [75](#)
- Cairo::FontOptions, [75](#)
  - ~FontOptions, [78](#)
  - cobj, [78](#)
  - cobject, [77](#)

- DEFAULT, 77
- FontOptions, 78
- FULL, 77
- get\_antialias, 78
- get\_hint\_metrics, 78
- get\_hint\_style, 79
- get\_subpixel\_order, 79
- hash, 79
- HintMetrics, 77
- HintStyle, 77
- m\_cobject, 81
- MEDIUM, 77
- merge, 79
- NONE, 77
- OFF, 77
- ON, 77
- operator=, 80
- operator==, 80
- set\_antialias, 80
- set\_hint\_metrics, 80
- set\_hint\_style, 80
- set\_subpixel\_order, 81
- SLIGHT, 77
- Cairo::FtFontFace, 81
  - create, 83
  - FtFontFace, 82
  - get\_synthesize, 83
  - set\_synthesize, 83
  - unset\_synthesize, 84
- Cairo::FtScaledFont, 84
  - create, 86
  - FtScaledFont, 86
  - lock\_face, 86
  - unlock\_face, 87
- Cairo::GlitzSurface, 87
  - ~GlitzSurface, 91
  - create, 91
  - GlitzSurface, 91
- Cairo::Gradient, 92
  - ~Gradient, 94
  - add\_color\_stop\_rgb, 94
  - add\_color\_stop\_rgba, 94
  - get\_color\_stops, 95
  - Gradient, 93, 94
- Cairo::ImageSurface, 96
  - ~ImageSurface, 100
  - create, 100
  - create\_from\_png, 101
  - create\_from\_png\_stream, 101
  - format\_stride\_for\_width, 102
  - get\_data, 102, 103
  - get\_format, 103
  - get\_height, 103
  - get\_stride, 103
  - get\_width, 103
  - ImageSurface, 100
- Cairo::LinearGradient, 104
  - ~LinearGradient, 106
  - create, 107
  - get\_linear\_points, 107
  - LinearGradient, 106
- Cairo::logic\_error, 108
  - ~logic\_error, 108
  - get\_status\_code, 109
  - logic\_error, 108
- Cairo::Matrix, 109
  - identity\_matrix, 113
  - invert, 111
  - Matrix, 110
  - multiply, 111
  - operator\*, 113
  - rotate, 112
  - rotation\_matrix, 114
  - scale, 112
  - scaling\_matrix, 114
  - transform\_distance, 112
  - transform\_point, 113
  - translate, 113
  - translation\_matrix, 114
- Cairo::Path, 115
  - ~Path, 116
  - cobj, 116
  - cobject, 115
  - m\_cobject, 116
  - operator=, 116
  - Path, 115
- Cairo::Pattern, 116
  - ~Pattern, 119
  - cobj, 119
  - cobject, 118
  - Extend, 118
  - get\_extend, 120
  - get\_matrix, 120
  - get\_type, 120
  - LINEAR, 119
  - m\_cobject, 122
  - NONE, 118
  - operator=, 120
  - PAD, 118
  - Pattern, 119
  - RADIAL, 119
  - reference, 120
  - REFLECT, 118
  - REPEAT, 118
  - set\_extend, 121
  - set\_matrix, 122
  - SOLID, 119
  - SURFACE, 119
  - Type, 118
  - unreference, 122
- Cairo::PdfSurface, 123
  - ~PdfSurface, 126
  - create, 127
  - create\_for\_stream, 127
  - get\_versions, 127
  - PdfSurface, 126

- restrict\_to\_version, 128
- set\_size, 128
- version\_to\_string, 128
- Cairo::PsSurface, 129
  - ~PsSurface, 133
  - create, 133
  - create\_for\_stream, 133
  - dsc\_begin\_page\_setup, 134
  - dsc\_begin\_setup, 134
  - dsc\_comment, 134
  - get\_eps, 135
  - get\_levels, 135
  - level\_to\_string, 135
  - PsSurface, 133
  - restrict\_to\_level, 135
  - set\_eps, 136
  - set\_size, 136
- Cairo::QuartzFontFace, 137
  - create, 138
  - QuartzFontFace, 138
- Cairo::QuartzSurface, 139
  - ~QuartzSurface, 143
  - create, 143
  - get\_cg\_context, 144
  - QuartzSurface, 142
- Cairo::RadialGradient, 144
  - ~RadialGradient, 147
  - create, 147
  - get\_radial\_circles, 147
  - RadialGradient, 146
- Cairo::RecordingSurface, 148
  - ~RecordingSurface, 152
  - create, 152
  - get\_extents, 153
  - ink\_extents, 153
  - RecordingSurface, 152
- Cairo::Region, 153
  - ~Region, 156
  - cobj, 156
  - cobject, 155
  - contains\_point, 156
  - contains\_rectangle, 156
  - copy, 156
  - create, 156, 157
  - do\_union, 157
  - do\_xor, 157
  - empty, 158
  - get\_extents, 158
  - get\_num\_rectangles, 158
  - get\_rectangle, 158
  - IN, 155
  - intersect, 158
  - m\_cobject, 159
  - OUT, 155
  - Overlap, 155
  - reference, 158
  - Region, 155
  - REGION\_OVERLAP\_PART, 155
  - subtract, 159
  - translate, 159
  - unreference, 159
- Cairo::SaveGuard, 159
  - ~SaveGuard, 160
  - SaveGuard, 160
- Cairo::ScaledFont, 161
  - ~ScaledFont, 163
  - cobj, 163
  - cobject, 162
  - create, 163
  - get\_ctm, 163
  - get\_extents, 164
  - get\_font\_face, 164
  - get\_font\_matrix, 164
  - get\_font\_options, 164
  - get\_glyph\_extents, 165
  - get\_scale\_matrix, 165
  - get\_text\_extents, 165
  - get\_type, 166
  - m\_cobject, 167
  - operator=, 166
  - ScaledFont, 162, 163
  - text\_to\_glyphs, 166
- Cairo::SolidPattern, 167
  - ~SolidPattern, 169
  - create\_rgb, 169
  - create\_rgba, 169
  - get\_rgba, 170
  - SolidPattern, 169
- Cairo::Surface, 171
  - ~Surface, 178
  - A1, 175
  - A8, 175
  - ARGB32, 175
  - BEOS, 176
  - cobj, 178
  - cobject, 174
  - copy\_page, 178
  - create, 178, 180
  - DIRECTFB, 176
  - DRM, 177
  - finish, 180
  - flush, 180
  - Format, 175
  - get\_content, 181
  - get\_device, 181
  - get\_device\_offset, 181
  - get\_device\_scale, 181
  - get\_fallback\_resolution, 182
  - get\_font\_options, 182
  - get\_mime\_data, 182
  - get\_type, 183
  - GL, 177
  - GLITZ, 176
  - has\_show\_text\_glyphs, 183
  - IMAGE, 176
  - m\_cobject, 187

- mark\_dirty, 183
- operator=, 184
- OS2, 176
- PDF, 176
- PS, 176
- QT, 176
- QUARTZ, 176
- QUARTZ\_IMAGE, 176
- RECORDING, 177
- RGB16\_565, 175
- RGB24, 175
- SCRIPT, 176
- set\_device\_offset, 184
- set\_device\_scale, 184
- set\_fallback\_resolution, 185
- set\_mime\_data, 185
- show\_page, 186
- SKIA, 177
- SlotDestroy, 174
- SlotReadFunc, 174
- SlotWriteFunc, 175
- SUBSURFACE, 177
- Surface, 177, 178
- SVG, 176
- TEE, 177
- Type, 175
- unset\_mime\_data, 186
- VG, 177
- WIN32, 176
- WIN32\_PRINTING, 176
- WIN32\_SURFACE, 176
- write\_to\_png, 186
- write\_to\_png\_stream, 187
- XCB, 176
- XLIB, 176
- XML, 177
- Cairo::SurfacePattern, 188
  - ~SurfacePattern, 191
  - BEST, 190
  - BILINEAR, 190
  - create, 191
  - FAST, 190
  - Filter, 190
  - GAUSSIAN, 190
  - get\_filter, 191
  - get\_surface, 191
  - GOOD, 190
  - NEAREST, 190
  - set\_filter, 191
  - SurfacePattern, 190
- Cairo::SvgSurface, 192
  - ~SvgSurface, 195
  - create, 196
  - create\_for\_stream, 196
  - get\_versions, 196
  - restrict\_to\_version, 197
  - SvgSurface, 195
  - version\_to\_string, 197
- Cairo::ToyFontFace, 198
  - BOLD, 200
  - create, 200
  - get\_family, 200
  - get\_slant, 201
  - get\_weight, 201
  - ITALIC, 200
  - NORMAL, 200
  - OBLIQUE, 200
  - Slant, 199
  - ToyFontFace, 200
  - Weight, 200
- Cairo::UserFontFace, 201
  - ~UserFontFace, 203
  - init, 204
  - render\_glyph, 204
  - text\_to\_glyphs, 205
  - unicode\_to\_glyph, 206
  - UserFontFace, 203
- Cairo::Win32FontFace, 208
  - create, 209, 210
  - Win32FontFace, 209
- Cairo::Win32PrintingSurface, 211
  - ~Win32PrintingSurface, 214
  - create, 214
  - Win32PrintingSurface, 214
- Cairo::Win32ScaledFont, 215
  - create, 217
  - done\_font, 217
  - get\_device\_to\_logical, 217
  - get\_logical\_to\_device, 218
  - get\_metrics\_factor, 218
  - select\_font, 218
  - Win32ScaledFont, 217
- Cairo::Win32Surface, 219
  - ~Win32Surface, 223
  - create, 223
  - create\_with\_ddb, 223
  - create\_with\_dib, 223
  - get\_dc, 224
  - get\_image, 224
  - Win32Surface, 222
- Cairo::XlibSurface, 225
  - ~XlibSurface, 229
  - create, 229
  - create\_with\_xrender\_format, 230
  - get\_depth, 230
  - get\_display, 230, 231
  - get\_drawable, 231
  - get\_height, 231
  - get\_screen, 231
  - get\_visual, 231
  - get\_width, 231
  - get\_xrender\_format, 232
  - set\_drawable, 232
  - set\_size, 232
  - XlibSurface, 228
- cairo\_matrix\_t, 233

- Cairomm: A C++ wrapper for the cairo graphics library, [1](#)
- CLEAR
  - Cairo::Context, [30](#)
- clip
  - Cairo::Context, [32](#)
- clip\_preserve
  - Cairo::Context, [33](#)
- close\_path
  - Cairo::Context, [33](#)
- cobj
  - Cairo::Context, [33](#), [34](#)
  - Cairo::Device, [70](#)
  - Cairo::FontFace, [74](#)
  - Cairo::FontOptions, [78](#)
  - Cairo::Path, [116](#)
  - Cairo::Pattern, [119](#)
  - Cairo::Region, [156](#)
  - Cairo::ScaledFont, [163](#)
  - Cairo::Surface, [178](#)
- cobject
  - Cairo::Context, [28](#)
  - Cairo::Device, [69](#)
  - Cairo::FontFace, [74](#)
  - Cairo::FontOptions, [77](#)
  - Cairo::Path, [115](#)
  - Cairo::Pattern, [118](#)
  - Cairo::Region, [155](#)
  - Cairo::ScaledFont, [162](#)
  - Cairo::Surface, [174](#)
- contains\_point
  - Cairo::Region, [156](#)
- contains\_rectangle
  - Cairo::Region, [156](#)
- Content
  - Cairo, [17](#)
- CONTENT\_ALPHA
  - Cairo, [17](#)
- CONTENT\_COLOR
  - Cairo, [17](#)
- CONTENT\_COLOR\_ALPHA
  - Cairo, [17](#)
- Context
  - Cairo::Context, [30](#)
- copy
  - Cairo::Region, [156](#)
- copy\_clip\_rectangle\_list
  - Cairo::Context, [34](#)
- copy\_page
  - Cairo::Context, [34](#)
  - Cairo::Surface, [178](#)
- copy\_path
  - Cairo::Context, [34](#)
- copy\_path\_flat
  - Cairo::Context, [35](#)
- create
  - Cairo::Context, [35](#)
  - Cairo::FtFontFace, [83](#)
- Cairo::FtScaledFont, [86](#)
- Cairo::GlitzSurface, [91](#)
- Cairo::ImageSurface, [100](#)
- Cairo::LinearGradient, [107](#)
- Cairo::PdfSurface, [127](#)
- Cairo::PsSurface, [133](#)
- Cairo::QuartzFontFace, [138](#)
- Cairo::QuartzSurface, [143](#)
- Cairo::RadialGradient, [147](#)
- Cairo::RecordingSurface, [152](#)
- Cairo::Region, [156](#), [157](#)
- Cairo::ScaledFont, [163](#)
- Cairo::Surface, [178](#), [180](#)
- Cairo::SurfacePattern, [191](#)
- Cairo::SvgSurface, [196](#)
- Cairo::ToyFontFace, [200](#)
- Cairo::Win32FontFace, [209](#), [210](#)
- Cairo::Win32PrintingSurface, [214](#)
- Cairo::Win32ScaledFont, [217](#)
- Cairo::Win32Surface, [223](#)
- Cairo::XlibSurface, [229](#)
- create\_for\_stream
  - Cairo::PdfSurface, [127](#)
  - Cairo::PsSurface, [133](#)
  - Cairo::SvgSurface, [196](#)
- create\_from\_png
  - Cairo::ImageSurface, [101](#)
- create\_from\_png\_stream
  - Cairo::ImageSurface, [101](#)
- create\_rgb
  - Cairo::SolidPattern, [169](#)
- create\_rgba
  - Cairo::SolidPattern, [169](#)
- create\_with\_ddb
  - Cairo::Win32Surface, [223](#)
- create\_with\_dib
  - Cairo::Win32Surface, [223](#)
- create\_with\_xrender\_format
  - Cairo::XlibSurface, [230](#)
- curve\_to
  - Cairo::Context, [35](#)
- DEFAULT
  - Cairo::FontOptions, [77](#)
- Deprecated List, [5](#)
- DEST
  - Cairo::Context, [30](#)
- DEST\_ATOP
  - Cairo::Context, [30](#)
- DEST\_IN
  - Cairo::Context, [30](#)
- DEST\_OUT
  - Cairo::Context, [30](#)
- DEST\_OVER
  - Cairo::Context, [30](#)
- Device
  - Cairo::Device, [69](#)
- device\_to\_user
  - Cairo::Context, [36](#)

- device\_to\_user\_distance
  - Cairo::Context, [36](#)
- DeviceType
  - Cairo::Device, [69](#)
- DIRECTFB
  - Cairo::Surface, [176](#)
- do\_union
  - Cairo::Region, [157](#)
- do\_xor
  - Cairo::Region, [157](#)
- done\_font
  - Cairo::Win32ScaledFont, [217](#)
- DRM
  - Cairo::Device, [69](#)
  - Cairo::Surface, [177](#)
- dsc\_begin\_page\_setup
  - Cairo::PsSurface, [134](#)
- dsc\_begin\_setup
  - Cairo::PsSurface, [134](#)
- dsc\_comment
  - Cairo::PsSurface, [134](#)
- empty
  - Cairo::Region, [158](#)
- EVEN\_ODD
  - Cairo::Context, [29](#)
- Extend
  - Cairo::Pattern, [118](#)
- FAST
  - Cairo::SurfacePattern, [190](#)
- fill
  - Cairo::Context, [36](#)
- fill\_preserve
  - Cairo::Context, [36](#)
- FillRule
  - Cairo::Context, [28](#)
- Filter
  - Cairo::SurfacePattern, [190](#)
- finish
  - Cairo::Device, [70](#)
  - Cairo::Surface, [180](#)
- flush
  - Cairo::Device, [70](#)
  - Cairo::Surface, [180](#)
- FONT\_TYPE\_FT
  - Cairo, [18](#)
- FONT\_TYPE\_QUARTZ
  - Cairo, [18](#)
- FONT\_TYPE\_TOY
  - Cairo, [18](#)
- FONT\_TYPE\_USER
  - Cairo, [18](#)
- FONT\_TYPE\_WIN32
  - Cairo, [18](#)
- FontExtents
  - Cairo, [16](#)
- FontFace
  - Cairo::FontFace, [74](#)
- FontOptions
  - Cairo::FontOptions, [78](#)
- FontType
  - Cairo, [17](#)
- Format
  - Cairo::Surface, [175](#)
- format\_stride\_for\_width
  - Cairo::ImageSurface, [102](#)
- FT\_SYNTHESIZE\_BOLT
  - Cairo, [18](#)
- FT\_SYNTHESIZE\_OBLIQUE
  - Cairo, [18](#)
- FtFontFace
  - Cairo::FtFontFace, [82](#)
- FtScaledFont
  - Cairo::FtScaledFont, [86](#)
- FtSynthesize
  - Cairo, [18](#)
- FULL
  - Cairo::FontOptions, [77](#)
- GAUSSIAN
  - Cairo::SurfacePattern, [190](#)
- get\_antialias
  - Cairo::Context, [37](#)
  - Cairo::FontOptions, [78](#)
- get\_cg\_context
  - Cairo::QuartzSurface, [144](#)
- get\_clip\_extents
  - Cairo::Context, [37](#)
- get\_color\_stops
  - Cairo::Gradient, [95](#)
- get\_content
  - Cairo::Surface, [181](#)
- get\_ctm
  - Cairo::ScaledFont, [163](#)
- get\_current\_point
  - Cairo::Context, [37](#)
- get\_dash
  - Cairo::Context, [38](#)
- get\_data
  - Cairo::ImageSurface, [102](#), [103](#)
- get\_dc
  - Cairo::Win32Surface, [224](#)
- get\_depth
  - Cairo::XlibSurface, [230](#)
- get\_device
  - Cairo::Surface, [181](#)
- get\_device\_offset
  - Cairo::Surface, [181](#)
- get\_device\_scale
  - Cairo::Surface, [181](#)
- get\_device\_to\_logical
  - Cairo::Win32ScaledFont, [217](#)
- get\_display
  - Cairo::XlibSurface, [230](#), [231](#)
- get\_drawable
  - Cairo::XlibSurface, [231](#)
- get\_eps

- Cairo::PsSurface, 135
- get\_extend
  - Cairo::Pattern, 120
- get\_extents
  - Cairo::RecordingSurface, 153
  - Cairo::Region, 158
  - Cairo::ScaledFont, 164
- get\_fallback\_resolution
  - Cairo::Surface, 182
- get\_family
  - Cairo::ToyFontFace, 200
- get\_fill\_extents
  - Cairo::Context, 38
- get\_fill\_rule
  - Cairo::Context, 38
- get\_filter
  - Cairo::SurfacePattern, 191
- get\_font\_extents
  - Cairo::Context, 39
- get\_font\_face
  - Cairo::Context, 39
  - Cairo::ScaledFont, 164
- get\_font\_matrix
  - Cairo::Context, 39
  - Cairo::ScaledFont, 164
- get\_font\_options
  - Cairo::Context, 39
  - Cairo::ScaledFont, 164
  - Cairo::Surface, 182
- get\_format
  - Cairo::ImageSurface, 103
- get\_glyph\_extents
  - Cairo::Context, 40
  - Cairo::ScaledFont, 165
- get\_group\_target
  - Cairo::Context, 40
- get\_height
  - Cairo::ImageSurface, 103
  - Cairo::XlibSurface, 231
- get\_hint\_metrics
  - Cairo::FontOptions, 78
- get\_hint\_style
  - Cairo::FontOptions, 79
- get\_image
  - Cairo::Win32Surface, 224
- get\_levels
  - Cairo::PsSurface, 135
- get\_line\_cap
  - Cairo::Context, 41
- get\_line\_join
  - Cairo::Context, 41
- get\_line\_width
  - Cairo::Context, 41
- get\_linear\_points
  - Cairo::LinearGradient, 107
- get\_logical\_to\_device
  - Cairo::Win32ScaledFont, 218
- get\_matrix
  - Cairo::Context, 41
  - Cairo::Pattern, 120
- get\_metrics\_factor
  - Cairo::Win32ScaledFont, 218
- get\_mime\_data
  - Cairo::Surface, 182
- get\_miter\_limit
  - Cairo::Context, 42
- get\_num\_rectangles
  - Cairo::Region, 158
- get\_operator
  - Cairo::Context, 42
- get\_path\_extents
  - Cairo::Context, 42
- get\_radial\_circles
  - Cairo::RadialGradient, 147
- get\_rectangle
  - Cairo::Region, 158
- get\_rgba
  - Cairo::SolidPattern, 170
- get\_scale\_matrix
  - Cairo::ScaledFont, 165
- get\_scaled\_font
  - Cairo::Context, 42
- get\_screen
  - Cairo::XlibSurface, 231
- get\_slant
  - Cairo::ToyFontFace, 201
- get\_source
  - Cairo::Context, 43
- get\_source\_for\_surface
  - Cairo::Context, 43
- get\_status\_code
  - Cairo::logic\_error, 109
- get\_stride
  - Cairo::ImageSurface, 103
- get\_stroke\_extents
  - Cairo::Context, 43
- get\_subpixel\_order
  - Cairo::FontOptions, 79
- get\_surface
  - Cairo::SurfacePattern, 191
- get\_synthesize
  - Cairo::FtFontFace, 83
- get\_target
  - Cairo::Context, 44
- get\_text\_extents
  - Cairo::Context, 44
  - Cairo::ScaledFont, 165
- get\_tolerance
  - Cairo::Context, 45
- get\_type
  - Cairo::Device, 71
  - Cairo::FontFace, 75
  - Cairo::Pattern, 120
  - Cairo::ScaledFont, 166
  - Cairo::Surface, 183
- get\_versions

- Cairo::PdfSurface, [127](#)
  - Cairo::SvgSurface, [196](#)
- get\_visual
  - Cairo::XlibSurface, [231](#)
- get\_weight
  - Cairo::ToyFontFace, [201](#)
- get\_width
  - Cairo::ImageSurface, [103](#)
  - Cairo::XlibSurface, [231](#)
- get\_xrender\_format
  - Cairo::XlibSurface, [232](#)
- GL
  - Cairo::Device, [69](#)
  - Cairo::Surface, [177](#)
- GLITZ
  - Cairo::Surface, [176](#)
- GlitzSurface
  - Cairo::GlitzSurface, [91](#)
- Glyph
  - Cairo, [16](#)
- glyph\_path
  - Cairo::Context, [45](#)
- GOOD
  - Cairo::SurfacePattern, [190](#)
- Gradient
  - Cairo::Gradient, [93](#), [94](#)
- green
  - Cairo::ColorStop, [21](#)
- has\_current\_point
  - Cairo::Context, [45](#)
- has\_show\_text\_glyphs
  - Cairo::Surface, [183](#)
- hash
  - Cairo::FontOptions, [79](#)
- hash\_load\_check\_resize\_trigger\_size\_base, [233](#)
- HintMetrics
  - Cairo::FontOptions, [77](#)
- HintStyle
  - Cairo::FontOptions, [77](#)
- identity\_matrix
  - Cairo::Matrix, [113](#)
- IMAGE
  - Cairo::Surface, [176](#)
- ImageSurface
  - Cairo::ImageSurface, [100](#)
- IN
  - Cairo::Context, [30](#)
  - Cairo::Region, [155](#)
- in\_clip
  - Cairo::Context, [45](#)
- in\_fill
  - Cairo::Context, [46](#)
- in\_stroke
  - Cairo::Context, [46](#)
- init
  - Cairo::UserFontFace, [204](#)
- ink\_extents
  - Cairo::RecordingSurface, [153](#)
- intersect
  - Cairo::Region, [158](#)
- invert
  - Cairo::Matrix, [111](#)
- ITALIC
  - Cairo::ToyFontFace, [200](#)
- level\_to\_string
  - Cairo::PsSurface, [135](#)
- line\_to
  - Cairo::Context, [47](#)
- LINEAR
  - Cairo::Pattern, [119](#)
- LinearGradient
  - Cairo::LinearGradient, [106](#)
- LineCap
  - Cairo::Context, [29](#)
- LineJoin
  - Cairo::Context, [29](#)
- Lock
  - Cairo::Device::Lock, [72](#)
- lock\_face
  - Cairo::FtScaledFont, [86](#)
- logic\_error
  - Cairo::logic\_error, [108](#)
- lu\_counter\_policy\_base, [234](#)
- m\_cobject
  - Cairo::Context, [67](#)
  - Cairo::Device, [71](#)
  - Cairo::FontFace, [75](#)
  - Cairo::FontOptions, [81](#)
  - Cairo::Path, [116](#)
  - Cairo::Pattern, [122](#)
  - Cairo::Region, [159](#)
  - Cairo::ScaledFont, [167](#)
  - Cairo::Surface, [187](#)
- make\_refptr\_for\_instance
  - Cairo, [20](#)
- mark\_dirty
  - Cairo::Surface, [183](#)
- mask
  - Cairo::Context, [47](#)
- mask\_based\_range\_hashing, [234](#)
- Matrix
  - Cairo::Matrix, [110](#)
- MEDIUM
  - Cairo::FontOptions, [77](#)
- merge
  - Cairo::FontOptions, [79](#)
- MITER
  - Cairo::Context, [29](#)
- mod\_based\_range\_hashing, [235](#)
- move\_to
  - Cairo::Context, [48](#)
- multiply
  - Cairo::Matrix, [111](#)



- NEAREST
  - Cairo::SurfacePattern, [190](#)
- New API in cairomm 1.18, [3](#)
- NONE
  - Cairo::FontOptions, [77](#)
  - Cairo::Pattern, [118](#)
- NORMAL
  - Cairo::ToyFontFace, [200](#)
- OBLIQUE
  - Cairo::ToyFontFace, [200](#)
- OFF
  - Cairo::FontOptions, [77](#)
- offset
  - Cairo::ColorStop, [21](#)
- ON
  - Cairo::FontOptions, [77](#)
- Operator
  - Cairo::Context, [29](#)
- operator=
  - Cairo::Context, [48](#)
  - Cairo::FontFace, [75](#)
  - Cairo::FontOptions, [80](#)
  - Cairo::Path, [116](#)
  - Cairo::Pattern, [120](#)
  - Cairo::ScaledFont, [166](#)
  - Cairo::Surface, [184](#)
- operator==
  - Cairo::FontOptions, [80](#)
- operator&
  - Cairo, [20](#)
- operator\*
  - Cairo::Matrix, [113](#)
- operator |
  - Cairo, [20](#)
- OS2
  - Cairo::Surface, [176](#)
- OUT
  - Cairo::Context, [30](#)
  - Cairo::Region, [155](#)
- OVER
  - Cairo::Context, [30](#)
- Overlap
  - Cairo::Region, [155](#)
- PAD
  - Cairo::Pattern, [118](#)
- paint
  - Cairo::Context, [48](#)
- paint\_with\_alpha
  - Cairo::Context, [48](#)
- Path
  - Cairo::Path, [115](#)
- Pattern
  - Cairo::Pattern, [119](#)
- PDF
  - Cairo::Surface, [176](#)
- PDF\_VERSION\_1\_4
  - Cairo, [18](#)
- PDF\_VERSION\_1\_5
  - Cairo, [18](#)
- PdfSurface
  - Cairo::PdfSurface, [126](#)
- PdfVersion
  - Cairo, [18](#)
- pop\_group
  - Cairo::Context, [48](#)
- pop\_group\_to\_source
  - Cairo::Context, [49](#)
- PS
  - Cairo::Surface, [176](#)
- PS\_LEVEL\_2
  - Cairo, [19](#)
- PS\_LEVEL\_3
  - Cairo, [19](#)
- PsLevel
  - Cairo, [18](#)
- PsSurface
  - Cairo::PsSurface, [133](#)
- push\_group
  - Cairo::Context, [49](#)
- push\_group\_with\_content
  - Cairo::Context, [50](#)
- QT
  - Cairo::Surface, [176](#)
- QUARTZ
  - Cairo::Surface, [176](#)
- QUARTZ\_IMAGE
  - Cairo::Surface, [176](#)
- QuartzFontFace
  - Cairo::QuartzFontFace, [138](#)
- QuartzSurface
  - Cairo::QuartzSurface, [142](#)
- RADIAL
  - Cairo::Pattern, [119](#)
- RadialGradient
  - Cairo::RadialGradient, [146](#)
- RECORDING
  - Cairo::Surface, [177](#)
- RecordingSurface
  - Cairo::RecordingSurface, [152](#)
- Rectangle
  - Cairo, [16](#)
- rectangle
  - Cairo::Context, [50](#)
- RectangleInt
  - Cairo, [16](#)
- red
  - Cairo::ColorStop, [21](#)
- reference
  - Cairo::Device, [71](#)
  - Cairo::FontFace, [75](#)
  - Cairo::Pattern, [120](#)
  - Cairo::Region, [158](#)
- REFLECT
  - Cairo::Pattern, [118](#)

- RefPtr
  - Cairo, 16
- Region
  - Cairo::Region, 155
- REGION\_OVERLAP\_PART
  - Cairo::Region, 155
- rel\_curve\_to
  - Cairo::Context, 50
- rel\_line\_to
  - Cairo::Context, 51
- rel\_move\_to
  - Cairo::Context, 51
- release
  - Cairo::Device, 71
- render\_glyph
  - Cairo::UserFontFace, 204
- REPEAT
  - Cairo::Pattern, 118
- reset\_clip
  - Cairo::Context, 52
- restore
  - Cairo::Context, 52
- restrict\_to\_level
  - Cairo::PsSurface, 135
- restrict\_to\_version
  - Cairo::PdfSurface, 128
  - Cairo::SvgSurface, 197
- RGB16\_565
  - Cairo::Surface, 175
- RGB24
  - Cairo::Surface, 175
- rotate
  - Cairo::Context, 52
  - Cairo::Matrix, 112
- rotate\_degrees
  - Cairo::Context, 53
- rotation\_matrix
  - Cairo::Matrix, 114
- ROUND
  - Cairo::Context, 29
- SATURATE
  - Cairo::Context, 30
- save
  - Cairo::Context, 53
- SaveGuard
  - Cairo::SaveGuard, 160
- scale
  - Cairo::Context, 53
  - Cairo::Matrix, 112
- ScaledFont
  - Cairo::ScaledFont, 162, 163
- scaling\_matrix
  - Cairo::Matrix, 114
- SCRIPT
  - Cairo::Device, 69
  - Cairo::Surface, 176
- select\_font
  - Cairo::Win32ScaledFont, 218
- select\_font\_face
  - Cairo::Context, 53
- set\_antialias
  - Cairo::Context, 54
  - Cairo::FontOptions, 80
- set\_dash
  - Cairo::Context, 55
- set\_device\_offset
  - Cairo::Surface, 184
- set\_device\_scale
  - Cairo::Surface, 184
- set\_drawable
  - Cairo::XlibSurface, 232
- set\_eps
  - Cairo::PsSurface, 136
- set\_extend
  - Cairo::Pattern, 121
- set\_fallback\_resolution
  - Cairo::Surface, 185
- set\_fill\_rule
  - Cairo::Context, 55
- set\_filter
  - Cairo::SurfacePattern, 191
- set\_font\_face
  - Cairo::Context, 56
- set\_font\_matrix
  - Cairo::Context, 56
- set\_font\_options
  - Cairo::Context, 56
- set\_font\_size
  - Cairo::Context, 56
- set\_hint\_metrics
  - Cairo::FontOptions, 80
- set\_hint\_style
  - Cairo::FontOptions, 80
- set\_identity\_matrix
  - Cairo::Context, 58
- set\_line\_cap
  - Cairo::Context, 58
- set\_line\_join
  - Cairo::Context, 58
- set\_line\_width
  - Cairo::Context, 58
- set\_matrix
  - Cairo::Context, 59
  - Cairo::Pattern, 122
- set\_mime\_data
  - Cairo::Surface, 185
- set\_miter\_limit
  - Cairo::Context, 59
- set\_operator
  - Cairo::Context, 60
- set\_scaled\_font
  - Cairo::Context, 60
- set\_size
  - Cairo::PdfSurface, 128
  - Cairo::PsSurface, 136
  - Cairo::XlibSurface, 232

- set\_source
  - Cairo::Context, [60](#), [61](#)
- set\_source\_rgb
  - Cairo::Context, [61](#)
- set\_source\_rgba
  - Cairo::Context, [62](#)
- set\_subpixel\_order
  - Cairo::FontOptions, [81](#)
- set\_synthesize
  - Cairo::FtFontFace, [83](#)
- set\_tolerance
  - Cairo::Context, [62](#)
- show\_glyphs
  - Cairo::Context, [63](#)
- show\_page
  - Cairo::Context, [63](#)
  - Cairo::Surface, [186](#)
- show\_text
  - Cairo::Context, [63](#)
- show\_text\_glyphs
  - Cairo::Context, [63](#)
- SKIA
  - Cairo::Surface, [177](#)
- Slant
  - Cairo::ToyFontFace, [199](#)
- SLIGHT
  - Cairo::FontOptions, [77](#)
- SlotDestroy
  - Cairo::Surface, [174](#)
- SlotReadFunc
  - Cairo::Surface, [174](#)
- SlotWriteFunc
  - Cairo::Surface, [175](#)
- SOLID
  - Cairo::Pattern, [119](#)
- SolidPattern
  - Cairo::SolidPattern, [169](#)
- SOURCE
  - Cairo::Context, [30](#)
- SQUARE
  - Cairo::Context, [29](#)
- stroke
  - Cairo::Context, [64](#)
- stroke\_preserve
  - Cairo::Context, [65](#)
- SUBPIXEL\_ORDER\_BGR
  - Cairo, [19](#)
- SUBPIXEL\_ORDER\_DEFAULT
  - Cairo, [19](#)
- SUBPIXEL\_ORDER\_RGB
  - Cairo, [19](#)
- SUBPIXEL\_ORDER\_VBGR
  - Cairo, [19](#)
- SUBPIXEL\_ORDER\_VRGB
  - Cairo, [19](#)
- SubpixelOrder
  - Cairo, [19](#)
- SUBSURFACE
  - Cairo::Surface, [177](#)
- subtract
  - Cairo::Region, [159](#)
- SURFACE
  - Cairo::Pattern, [119](#)
- Surface
  - Cairo::Surface, [177](#), [178](#)
- SurfacePattern
  - Cairo::SurfacePattern, [190](#)
- SVG
  - Cairo::Surface, [176](#)
- SVG\_VERSION\_1\_1
  - Cairo, [19](#)
- SVG\_VERSION\_1\_2
  - Cairo, [19](#)
- SvgSurface
  - Cairo::SvgSurface, [195](#)
- SvgVersion
  - Cairo, [19](#)
- TEE
  - Cairo::Surface, [177](#)
- TEXT\_CLUSTER\_FLAG\_BACKWARD
  - Cairo, [19](#)
- text\_path
  - Cairo::Context, [65](#)
- text\_to\_glyphs
  - Cairo::ScaledFont, [166](#)
  - Cairo::UserFontFace, [205](#)
- TextCluster
  - Cairo, [16](#)
- TextClusterFlags
  - Cairo, [19](#)
- TextExtents
  - Cairo, [17](#)
- ToyFontFace
  - Cairo::ToyFontFace, [200](#)
- transform
  - Cairo::Context, [66](#)
- transform\_distance
  - Cairo::Matrix, [112](#)
- transform\_point
  - Cairo::Matrix, [113](#)
- translate
  - Cairo::Context, [66](#)
  - Cairo::Matrix, [113](#)
  - Cairo::Region, [159](#)
- translation\_matrix
  - Cairo::Matrix, [114](#)
- Type
  - Cairo::Pattern, [118](#)
  - Cairo::Surface, [175](#)
- unicode\_to\_glyph
  - Cairo::UserFontFace, [206](#)
- unlock\_face
  - Cairo::FtScaledFont, [87](#)
- unreference
  - Cairo::Device, [71](#)

- Cairo::FontFace, [75](#)
- Cairo::Pattern, [122](#)
- Cairo::Region, [159](#)
- unset\_dash
  - Cairo::Context, [66](#)
- unset\_mime\_data
  - Cairo::Surface, [186](#)
- unset\_synthesize
  - Cairo::FtFontFace, [84](#)
- user\_to\_device
  - Cairo::Context, [66](#)
- user\_to\_device\_distance
  - Cairo::Context, [67](#)
- UserFontFace
  - Cairo::UserFontFace, [203](#)
- version\_to\_string
  - Cairo::PdfSurface, [128](#)
  - Cairo::SvgSurface, [197](#)
- VG
  - Cairo::Surface, [177](#)
- Weight
  - Cairo::ToyFontFace, [200](#)
- WIN32
  - Cairo::Surface, [176](#)
- WIN32\_PRINTING
  - Cairo::Surface, [176](#)
- WIN32\_SURFACE
  - Cairo::Surface, [176](#)
- Win32FontFace
  - Cairo::Win32FontFace, [209](#)
- Win32PrintingSurface
  - Cairo::Win32PrintingSurface, [214](#)
- Win32ScaledFont
  - Cairo::Win32ScaledFont, [217](#)
- Win32Surface
  - Cairo::Win32Surface, [222](#)
- WINDING
  - Cairo::Context, [29](#)
- write\_to\_png
  - Cairo::Surface, [186](#)
- write\_to\_png\_stream
  - Cairo::Surface, [187](#)
- XCB
  - Cairo::Device, [69](#)
  - Cairo::Surface, [176](#)
- XLIB
  - Cairo::Device, [69](#)
  - Cairo::Surface, [176](#)
- XlibSurface
  - Cairo::XlibSurface, [228](#)
- XML
  - Cairo::Device, [69](#)
  - Cairo::Surface, [177](#)
- XOR
  - Cairo::Context, [30](#)