cairomm

1.18.0

Generated by Doxygen 1.12.0

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:

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

Chapter 6

Class Index

6.1 Class List

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

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_BOLT = 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).

- 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.

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

cairomm 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 taing 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_degres)

  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

```
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 Context::set_miter_limit()
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

cobject	The C instance.
has_reference	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

path	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

xc	X position of the center of the arc
yc	Y position of the center of the arc
radius	the radius of the arc
angle1	the start angle, in radians
angle2	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

xc	X position of the center of the arc
yc	Y position of the center of the arc
radius	the radius of the arc
angle1	the start angle, in radians
angle2	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

rectangles	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

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

x	X value of coordinate (in/out parameter)
y	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

dx	X component of a distance vector (in/out parameter)
dy	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

x1	left of the resulting extents
y1	top of the resulting extents
x2	right of the resulting extents
y2	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

x	return value for X coordinate of the current point
y	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

dashes	return value for the dash array.
offset	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

x1	left of the resulting extents
y1	top of the resulting extents
x2	right of the resulting extents
y2	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

extents	a Cairo::FontExtents 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

matrix	a Cairo::Matrix 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

options	a FontOptions 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

glyphs	a vector of glyphs
extents	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

matrix	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

x1	left of the resulting extents
y1	top of the resulting extents
x2	right of the resulting extents
y2	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

x1	left of the resulting extents
y1	top of the resulting extents
x2	right of the resulting extents
y2	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

utf8	a string of text encoded in UTF-8
extents	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

glyphs	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

x	X coordinate of the point to test
y	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

x	X coordinate of the point to test
y	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

x	X coordinate of the point to test
y	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

x	the X coordinate of the end of the new line
y	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

pattern	a Pattern

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

surface	a Surface
surface_x	X coordinate at which to place the origin of surface
surface_y	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

x	the X coordinate of the new position
y	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

alpha	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 their 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

content	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

x	the X coordinate of the top left corner of the rectangle
y	the Y coordinate to the top left corner of the rectangle
width	the width of the rectangle
height	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

dx1	the X offset to the first control point
dy1	the Y offset to the first control point
dx2	the X offset to the second control point
dy2	the Y offset to the second control point
dx3	the X offset to the end of the curve
dy3	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. Cairomm 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

dx	the X offset to the end of the new line
dy	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. Cairomm 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

dx	the X offset
dy	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

angle	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_degres)
```

A convenience wrapper around rotate() that accepts angles in degrees.

Parameters

angle_degrees	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

sx	scale factor for the X dimension
sy	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

family	a font family name, encoded in UTF-8
slant	the slant for the font
weight	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

antialias	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 altenate "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

dashes	an array specifying alternate lengths of on and off portions
offset	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

fill_rule	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 font_face*.

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

Parameters

font_face	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

matrix	a Cairo::Matrix 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

options	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

size	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

line_cap	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

line_join	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

width	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

matrix	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: miter_limit = 1/sin(angle/2)

Parameters

limit	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

op	a compositing operator, specified as a Operator

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

scaled_font	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

source	a Pattern 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

surface	a Surface to be used to set the source pattern
x	User-space X coordinate for surface origin
y	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

red	red component of color
green	green component of color
blue	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

red	red component of color
green	green component of color
blue	blue component of color
alpha	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

tolerance	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

glyphs	vector of glyphs to show
num_glyphs	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

utf8	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 <firstterm>clusters</firstterm>. 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

utf8	a string of text encoded in UTF-8
glyphs	vector of glyphs to show
clusters	vector of cluster mapping information
cluster_flags	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 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

utf8	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

matrix	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

tx	amount to translate in the X direction
ty	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

x	X value of coordinate (in/out parameter)
y	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

dx	X component of a distance vector (in/out parameter)
dy	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

```
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

cobject	The C instance.
has_reference	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:

- cairomm/device.h

8.4 Cairo::Device::Lock Class Reference

A convenience class for acquiring a Device object in an exception-safe manner.

```
#include <cairomm/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:

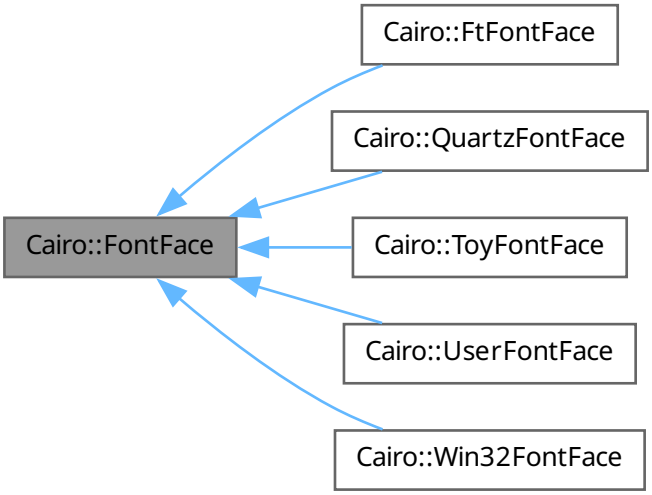
- cairomm/device.h

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

```
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

cobject	The C instance.
has_reference	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

```
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

other	another FontOptions

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

antialias	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

hint_metrics	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

hint_style	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

subpixel_order	the new subpixel order.

8.6.6 Member Data Documentation

8.6.6.1 m_cobject

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

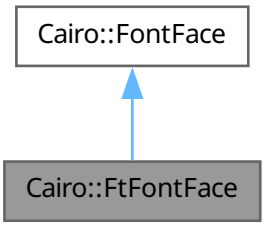
The documentation for this class was generated from the following file:

- 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 glypth synthesis options.
- FtSynthesize get_synthesize () const

  Returns currently activy 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

face	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 Cairo, the best way to determine when it is safe to free the face is to pass a cairo_destroy_func_t to cairo_font_face_set_user_data().
load_flags	flags to pass to FT_Load_Glyph when loading glyphs from the font. These flags are OR'ed together with the flags derived from the cairo_font_options_t passed to cairo_scaled_font_create(), so only a few values such as FT_LOAD_VERTICAL_LAYOUT, and FT_LOAD_FORCE_AUTOHINT are useful. You should not pass any of the flags affecting the load target, such as FT_LOAD_TARGET_LIGHT.

Since

1.8

8.7.2.2 get_synthesize()

```
FtSynthesize Cairo::FtFontFace::get_synthesize () const
```

Returns currently activy 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

synth_flags	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 glypth synthesis options.

Parameters

synth_flags	A set of synthesis options to disable

Since

1.12

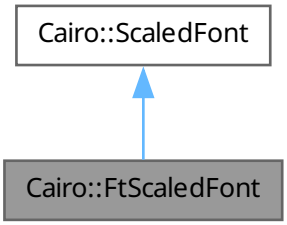
The documentation for this class was generated from the following file:

- 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)
- 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:

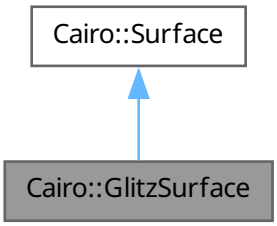
- cairomm/scaledfont.h

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

cobject	The C instance.
has_reference	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

surface	a glitz surface type

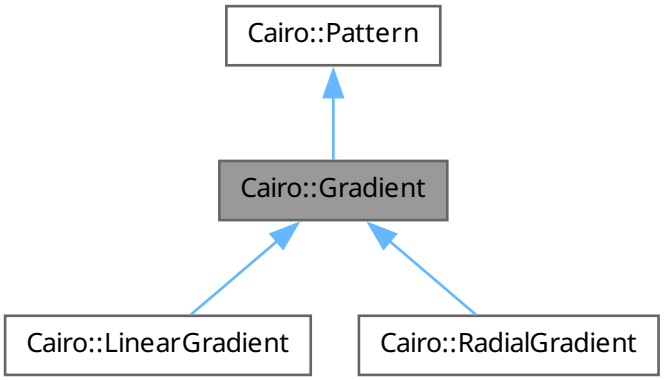
The documentation for this class was generated from the following file:

- 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

cobject	The C instance.
has_reference	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

offset	an offset in the range [0.0 .. 1.0]
red	red component of color
green	green component of color
blue	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

offset	an offset in the range [0.0 .. 1.0]
red	red component of color
green	green component of color
blue	blue component of color
alpha	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:

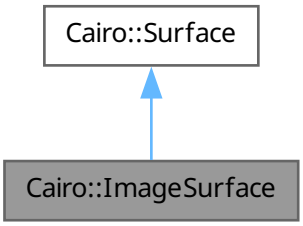
- cairomm/pattern.h

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 <cairomm/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

cobject	The C instance.
has_reference	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

format	format of pixels in the surface to create
width	width of the surface, in pixels
height	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

data	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).
format	the format of pixels in the buffer
width	the width of the image to be stored in the buffer
height	the height of the image to be stored in the buffer
stride	the number of bytes between the start of rows in the buffer as allocated. This value should always be computed by format_stride_for_width() 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

filename	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

read_func	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

format	A Format value
width	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.

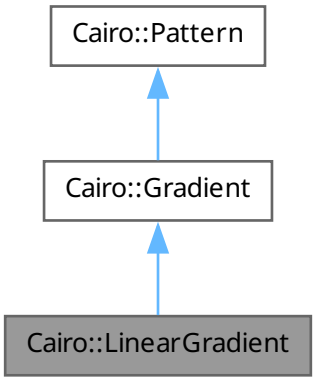
The documentation for this class was generated from the following file:

- 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

cobject	The C instance.
has_reference	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

x0	x coordinate of the start point
y0	y coordinate of the start point
x1	x coordinate of the end point
y1	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

x0	return value for the x coordinate of the first point
y0	return value for the y coordinate of the first point
x1	return value for the x coordinate of the second point
y1	return value for the y coordinate of the second point

Gets the gradient endpoints for a linear gradient.

Since

1.4

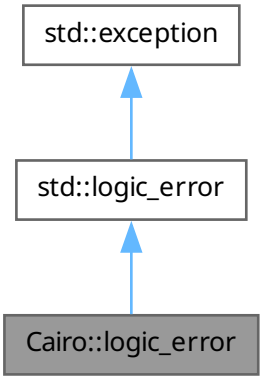
The documentation for this class was generated from the following file:

- cairomm/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

Public Member Functions inherited from std::exception

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:

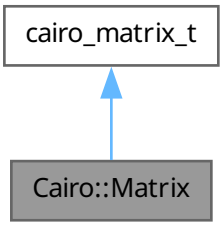
- cairomm/exception.h

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

a	a Matrix
b	a Matrix

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

radians	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

sx	scale factor in the X direction
sy	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 (x1,y1) transforms to (x2,y2) then (x1+dx1,y1+dy1) will transform to (x1+dx2,y1+dy2) for all values of x1 and x2.

Parameters

dx	X component of a distance vector. An in/out parameter
dy	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

x	X position. An in/out parameter
y	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

tx	amount to translate in the X direction
ty	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

a	a Matrix
b	a Matrix

8.14.4.3 rotation_matrix()

```
Matrix rotation_matrix (
            double radians)  [related]
```

Returns a Matrix initialized to a transformation that rotates by radians.

Parameters

radians	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

sx	scale factor in the X direction
sy	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

tx	amount to translate in the X direction
ty	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

```
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:

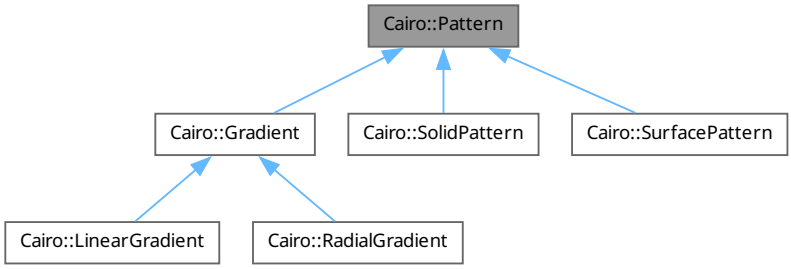
- cairomm/path.h

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

```
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 a 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

cobject	The C instance.
has_reference	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

Cairo::Extend	describing how the area outsize 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:

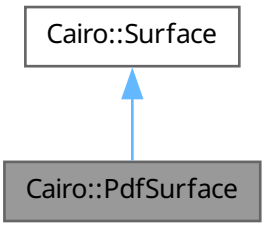
- cairomm/pattern.h

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

cobject	The C instance.
has_reference	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

filename	The name of the PDF file to save the surface to
width_in_points	The width of the PDF document in points
height_in_points	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

write_func	The function to be called when the backend needs to write data to an output stream
width_in_points	The width of the PDF document in points
height_in_points	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

width_in_points	new surface width, in points (1 point == 1/72.0 inch)
height_in_points	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:

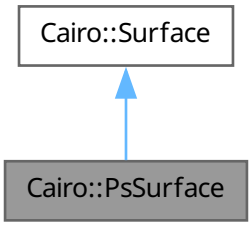
- cairomm/surface.h

8.18 Cairo::PsSurface Class Reference

A PsSurface provides a way to render PostScript documents from cairo.

```
#include <cairomm/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 PostSript 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

cobject	The C instance.
has_reference	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

filename	The name of the PostScript file to save the surface to
width_in_points	The width of the PostScript document in points
height_in_points	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

write_func	The function to be called when the backend needs to write data to an output stream
width_in_points	The width of the PostScript document in points
height_in_points	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

comment	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

level	a level id

Since

1.6

8.18.3.9 restrict_to_level()

```
void Cairo::PsSurface::restrict_to_level (
            PsLevel level)
```

Restricts the generated PostSript 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

level	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

width_in_points	new surface width, in points (1 point == 1/72.0 inch)
height_in_points	new surface height, in points (1 point == 1/72.0 inch)

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

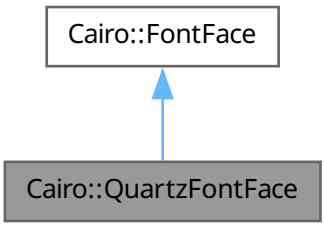
- cairomm/surface.h

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 (
            ATSUFontID font_id)  [protected]
```

8.19.3 Member Function Documentation

8.19.3.1 create() [1/2]

```
static RefPtr< QuartzFontFace > Cairo::QuartzFontFace::create (
            ATSUFontID font_id)  [static]
```

Creates a new font for the Quartz font backend based on an ATSUFontID.

This font can then be used with Context::set_font_face() or ScaledFont::create().

Parameters

font_id	an ATSUFontID 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

font	a CGFontRef obtained through a method external to cairo.

Since

1.8

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

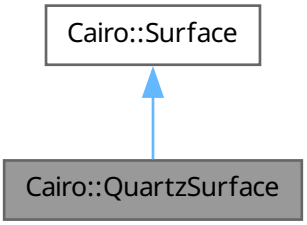
- cairomm/quartz_font.h

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

cobject	The C instance.
has_reference	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

cg_context	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

format	format of pixels in the surface to create
width	width of the surface, in pixels
height	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

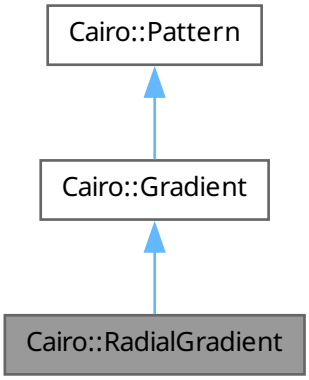
The documentation for this class was generated from the following file:

- 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

cobject	The C instance.
has_reference	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

cx0	x coordinate for the center of the start circle
cy0	y coordinate for the center of the start circle
radius0	radius of the start circle
cx1	x coordinate for the center of the end circle
cy1	y coordinate for the center of the end circle
radius1	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

x0	return value for the x coordinate of the center of the first (inner) circle
y0	return value for the y coordinate of the center of the first (inner) circle
r0	return value for the radius of the first (inner) circle
x1	return value for the x coordinate of the center of the second (outer) circle
y1	return value for the y coordinate of the center of the second (outer) circle
r1	return value for the radius of the second (outer) circle

Since

1.4

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

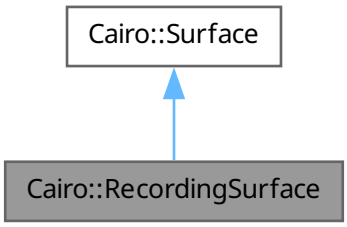
- cairomm/pattern.h

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

cobject	The C instance.
has_reference	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

extents	the extents to record
content	the content of the recording surface; defaults to Cairo::Content::CONTENT_COLOR_ALPHA.

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

content	the content of the recording surface; defaults to Cairo::Content::CONTENT_COLOR_ALPHA.

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

extents	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

```
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

cobject	The C instance.
has_reference	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 cairomm 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:

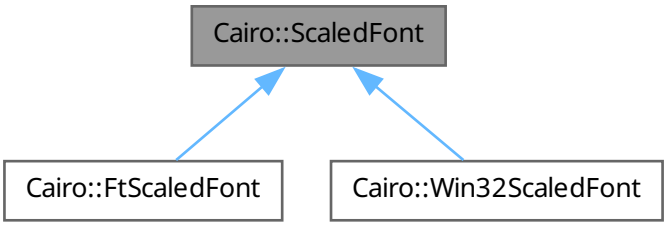
- cairomm/context.h

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

```
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

font_face	A font face.
font_matrix	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 Context::set_font_matrix().
ctm	user to device transformation matrix with which the font will be used.
options	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

glyphs	A vector of glyphs to calculate the extents of.
extents	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

scale_matrix	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

utf8	a string of text, encoded in UTF-8.
extents	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

x	X position to place first glyph.
y	Y position to place first glyph.
utf8	a string of text encoded in UTF-8.
glyphs	pointer to array of glyphs to fill.
clusters	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.

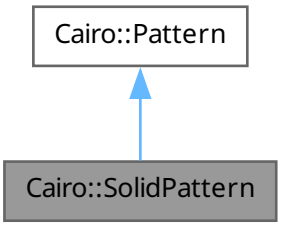
The documentation for this class was generated from the following file:

- 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

cobject	The C instance.
has_reference	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

red	red component of the color
green	green component of the color
blue	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

red	red component of the color
green	green component of the color
blue	blue component of the color
alpha	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

red	return value for red component of color
green	return value for green component of color
blue	return value for blue component of color
alpha	return value for alpha component of color

Since

1.4

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

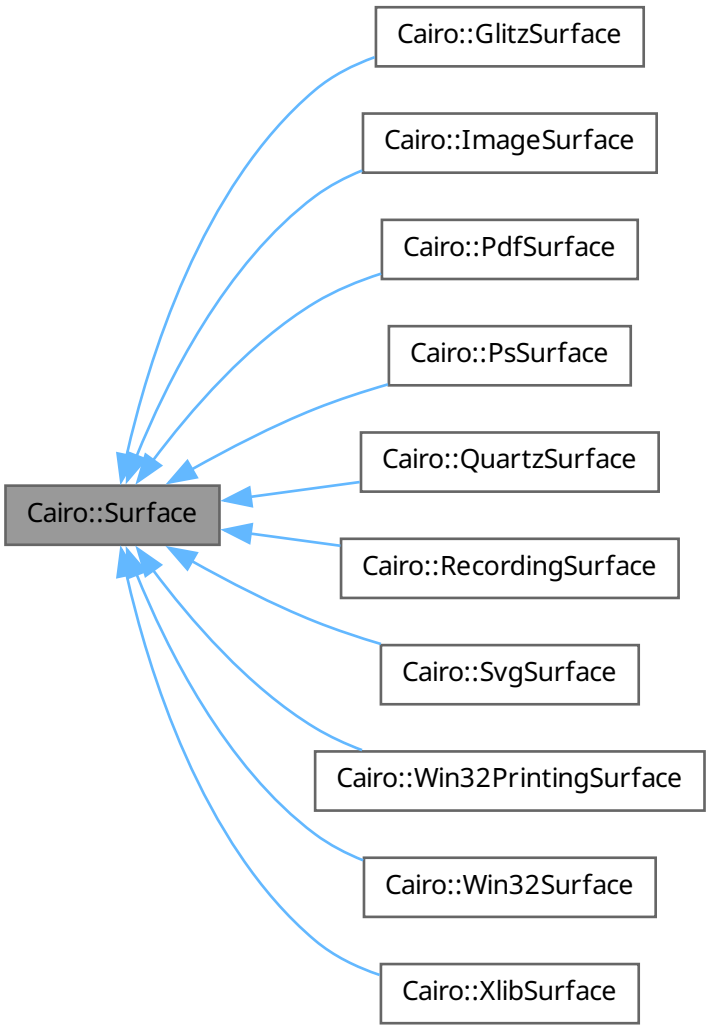
- cairomm/pattern.h

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 <cairomm/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

```
cairo_surface_t Cairo::Surface::cobject
```

The underlying C cairo surface type.

8.27.2.2 SlotDestroy

```
sigc::slot<void()> Cairo::Surface::SlotDestroy
```

For instance, void on_destroy();.

8.27.2.3 SlotReadFunc

```
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

data	the buffer into which to read the data
length	the amount of data to read

Returns

the status code of the read operation

8.27.2.4 SlotWriteFunc

```
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

data	the buffer containing the data to write
length	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 pikel is a 1-bit quentity holding an alpha value. Pixels are packed together into 32-bit quantities. The ordering of the bits matches the endianess 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 fixel 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 xlim.
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. **Deprecated** 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 Surface::create() **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

cobject	The C instance.
has_reference	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

target	an existing surface for which the sub-surface will point to
x	the x-origin of the sub-surface from the top-left of the target surface (in device-space units)
y	the y-origin of the sub-surface from the top-left of the target surface (in device-space units)
width	width of the sub-surface (in device-space units)
height	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

other	an existing surface used to select the backend of the new surface
content	the content for the new surface
width	width of the new surface, (in device-space units)
height	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

x_pixels_per_inch	horizontal pixels per inch
y_pixels_per_inch	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

options	a FontOptions 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

mime_type	The MIME type of the image data.
length	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

x	X coordinate of dirty rectangle
y	Y coordinate of dirty rectangle
width	width of dirty rectangle
height	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

x_offset	the offset in the X direction, in device units
y_offset	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

scale	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

x_scale	a scale factor in the X direction
y_scale	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 resoultion is 300 pixels per inch in both dimensions.

Parameters

x_pixels_per_inch	Pixels per inch in the x direction
y_pixels_per_inch	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

mime_type	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.
slot_destroy	A callback slot that will be called when the Surface no longer needs the data. For instance, when the Surface 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

filename	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

write_func	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.

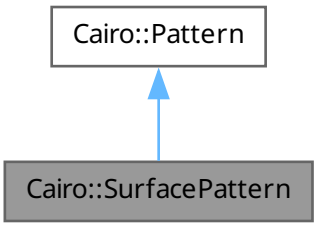
The documentation for this class was generated from the following file:

- cairomm/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 Cairo::Patern::Filter::NEAREST.
GOOD	A reasonable-performance filter, with quality similar to Cairo::BILINEAR.
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

cobject	The C instance.
has_reference	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

filter	Cairo::Filter describing the filter to use for resizing the pattern

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

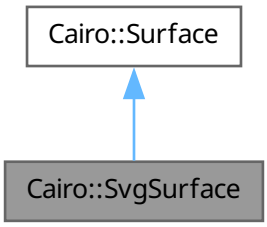
- cairomm/pattern.h

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

cobject	The C instance.
has_reference	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

filename	The name of the SVG file to save the surface to
width_in_points	The width of the SVG document in points
height_in_points	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

write_func	The function to be called when the backend needs to write data to an output stream
width_in_points	The width of the SVG document in points
height_in_points	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:

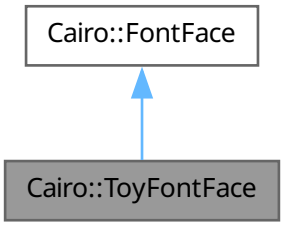
- cairomm/surface.h

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 familly 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

family	a font family name, encoded in UTF-8.
slant	the slant for the font.
weight	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 familly 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:

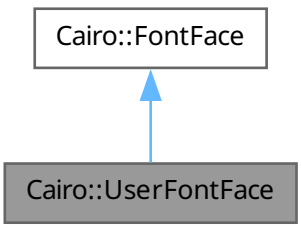
- cairomm/fontface.h

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.

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

scaled_font	the scaled-font being created
cr	cairo context, in font space
extents	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

scaled_font	user scaled-font
glyph	glyph code to render
cr	Context to draw to, in font space
extents	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.

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

scaled_font	the scaled-font being created
utf8	a string of text encoded in UTF-8
glyphs	array of glyphs to fill, in font space
clusters	array of cluster mapping information to fill
cluster_flags	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

scaled_font	the scaled-font being created
unicode	input unicode character code-point
glyph_index	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.

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

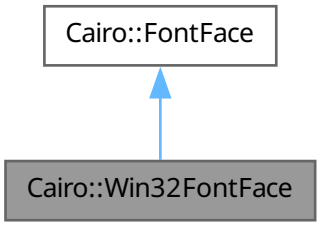
- cairomm/fontface.h

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

font	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

logfont	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

logfont	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.
font	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:

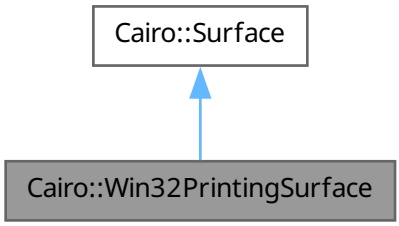
- cairomm/win32_font.h

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

hdc	the DC to create a surface for.

Since

1.8

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

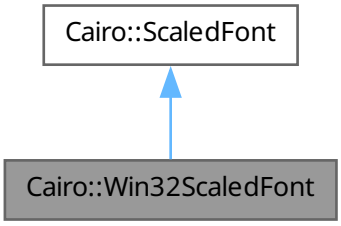
- cairomm/win32_surface.h

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

device_to_logical	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

logical_to_device	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

scaled_font	A cairo_scaled_font_t from the Win32 font backend. Such an object can be created with Win32FontFace::create_for_logfontw().
hdc	a device context

Since

1.8

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

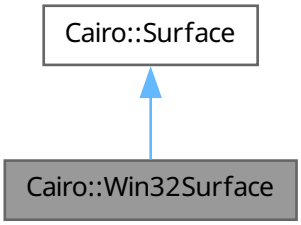
- cairomm/win32_font.h

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

cobject	The C instance.
has_reference	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

hdc	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

hdc	the DC to create a surface for.
format	format of pixels in the surface to create,
width	width of the surface, in pixels.
height	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 unititialized.

Parameters

format	format of pixels in the surface to create.
width	width of the surface, in pixels.
height	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:

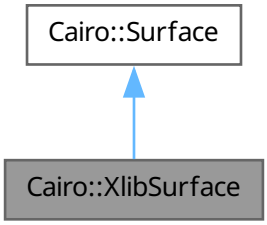
- cairomm/win32_surface.h

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 availabe, 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

cobject	The C instance.
has_reference	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

dpy	an X Display
drawable	an X Drawable, (a Pixmap or a Window)
visual	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.
width	the current width of drawable.
height	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

dpy	an X Display
bitmap	an X Drawable, (a depth-1 Pixmap)
screen	the X Screen associated with bitmap
width	the current width of bitmap.
height	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

dpy	an X Display
drawable	an X Drawable, (a Pixmap or a Window)
screen	the X Screen associated with @drawable
format	the picture format to use for drawing to @drawable. The depth of @format must match the depth of the drawable.
width	the current width of @drawable.
height	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

drawable	the new drawable for the surface
width	the width of the new drawable
height	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

width	the new width of the surface
height	the new height of the surface

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

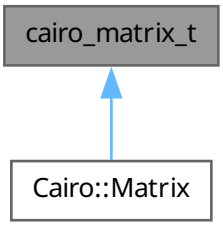
- cairomm/xlib_surface.h

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

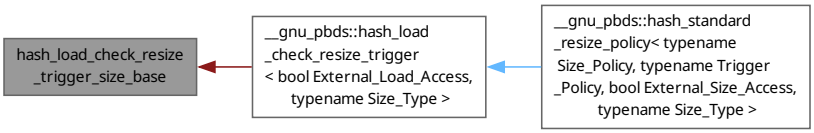
See the cairo_matrix_t reference in the cairo manual for more information.

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

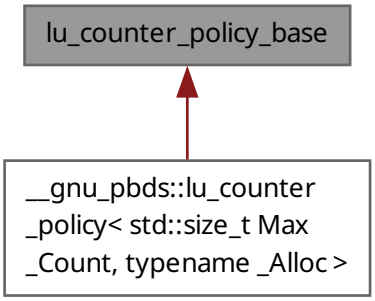
Inheritance diagram for hash_load_check_resize_trigger_size_base:



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

8.39 lu_counter_policy_base Class Reference

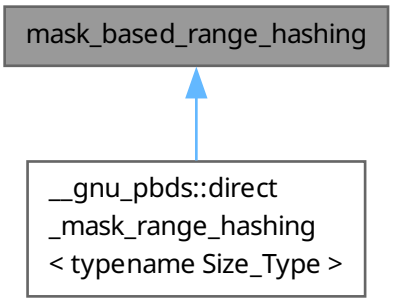
Inheritance diagram for lu_counter_policy_base:



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

8.40 mask_based_range_hashing Class Reference

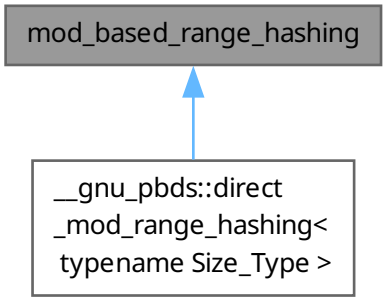
Inheritance diagram for mask_based_range_hashing:



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 an ineffienct 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