Document Number: Dxxxx
Date: 2014-11-06
Revises: N4073
Reply to: Michael B. McLaughlin
mikebmcl@gmail.com
Herb Sutter
Microsoft Inc.
hsutter@microsoft.com
Jason Zink
jzink_1@yahoo.com

Working Draft, Technical Specification — Graphics 2D

Note: this is an early draft. It's known to be incomplet and incorrekt, and it has lots of bad formatting.

Contents

List of Tables

List of Figures

1 General [io2d.general]

1.1 Scope [io2d.general.scope]

1 This Technical Specification specifies requirements for implementations of an interface that computer programs written in the C++ programming language may use to render and display raster graphics.

1.2 Normative references [io2d.general.refs]

1 The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

(1.1) — ISO/IEC 14882, *Programming languages — C++*

(1.2) — ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*

(1.3) — ISO/IEC TR 19769:2004, *Information technology — Programming languages, their environments and system software interfaces — Extensions for the programming language C to support new character data types*

(1.4) — ISO 32000-1:2008, *Document management — Portable document format — Part 1: PDF 1.7*

(1.5) — Tantek Çelik et al., *CSS Color Module Level 3 — W3C Recommendation 07 June 2011*, Copyright © 2011 W3C® (MIT, ERCIM, Keio)

(1.6) — Carl D. Worth, Chris Wilson, et al., *cairo graphics library — Version 1.12.16*, Copyright © 2002 University of Southern California, Copyright © 2005 Red Hat, Inc., Copyright © 2011 Intel Corporation

2 The library described in ISO/IEC TR 19769:2004 is hereinafter called the *C Unicode TR*.

3 The document CSS Color Module Level 3 — W3C Recommendation 07 June 2011 is hereinafter called the *CSS Colors Specification*.

4 Any section number reference to C++ 2014 or to C++ generally refers to the Working Draft in Document Number N4140.

1.3 Terms and definitions [io2d.general.defns]

1 For the purposes of this document, the following definitions apply.

1.3.1 [io2d.general.defns.alias]

aliasing

<aliasing> the presence of visual artifacts in the results of rendering due to sampling imperfections

1.3.2 [io2d.general.defns.anti.alias]

anti-aliasing

<anti-aliasing> the application of a function or algorithm while rendering to reduce aliasing

1.3.3 [io2d.general.defns.artifact]

artifact

<artifact> in signal processing, an error in the results due to the input signal and output signal not having a one-to-one mapping between their frequencies

1.3.4 **[io2d.general.defns.bezier]**

Bézier curve

<Bézier curve> a curve defined by a parametric equation where the starting point, ending point, and zero or more control points are given – a cubic Bézier curve, which is the type meant whenever the general term Bézier curve is used herein, has two control points and is defined by the following equation: $f(t) = (1-t)^3 \times P_0 + 3 \times t \times (1-t)^2 \times P_1 + 3 \times t^2 \times (1-t) \times P_2 + t^3 \times P_3$ where $0.0 \leq t \leq 1.0$, P_0 is the starting point, P_1 is the first control point, and P_2 is the second control point, and P_3 is the ending point [*Note:* $f(t) = P_0$ when $t = 0.0$ and $f(t) = P_3$ when $t = 1.0$ — *end note*]

1.3.5 **[io2d.general.defns.coordinate.space]**

coordinate space

<coordinate space> used to define specific positions on a surface; when a surface is displayed or otherwise rendered, the point (0.0, 0.0) is the top left, the positive X axis extends rightward, the positive Y axis extends downward, the point (1.0, 0.0) rotated around the point (0.0, 0.0) to 0 radians will be at (1.0, 0.0), the point (1.0, 0.0) rotated around the point (0.0, 0.0) to $\frac{(\pi)}{(2.0)}$ radians will be at (0.0, 1.0), the point (1.0, 0.0) rotated around the point (0.0, 0.0) to π radians will be at (-1.0, 0.0), the point (1.0, 0.0) rotated around the point (0.0, 0.0) to $\frac{(3.0 \times \pi)}{(2.0)}$ radians will be at (0.0, -1.0), and the point (1.0, 0.0) rotated around the point (0.0, 0.0) to $2.0 \times \pi$ radians will be at (0.0, 1.0)

1.3.6 **[io2d.general.defns.filter]**

filter

<filter> a mathematical function that transforms a signal into discrete data

1.3.7 **[io2d.general.defns.graphics.raster]**

graphics

<raster graphics> visual information stored as a rectangular grid of pixels

1.3.8 **[io2d.general.defns.graphics.vector]**

graphics

<vector graphics> visual information stored as geometrical data

1.3.9 **[io2d.general.defns.pixel]**

pixel

<pixel> a discrete data element with a format-dependent composition [*Note:* In raster graphics a pixel is obtained by sampling visual data, whether electro-mechanically, e.g. by a digital camera when a picture is taken, or manually, e.g. by an artist using a raster graphics editing program to create a pixel representation of visual data; the given examples typically result in a texture composed of many pixels — *end note*]

1.3.10 **[io2d.general.defns.point]**

point

<point> a coordinate designated by an X component and a Y component within the coordinate space of a surface

1.3.11 **[io2d.general.defns.render]**

render
<render> the process of compositing source data and destination data to create result data that replaces the destination data [*Note:* depending on the compositing operator and the clip, there may not be any change to the original destination data. — *end note*]

1.3.12 **[io2d.general.defns.sampling]**
sampling
<sampling> the transformation of continuous or discrete signals to discrete data by application of a filter

1.3.13 **[io2d.general.defns.signal.cont]**
signal
<continuous signal> a real function which is defined at every point within a 2D space and conveys information about one or more properties of the space

1.3.14 **[io2d.general.defns.signal]**
signal
<discrete signal> a real function which is defined at points separated by integer intervals within a 2D space and conveys information about one or more properties of the space

1.4 Implementation compliance [io2d.general.compliance]

1 The set of *diagnosable rules* consists of all syntactic and semantic rules in this International Standard except for those rules containing an explicit notation that "no diagnostic is required" or which are described as resulting in "undefined behavior."

1.5 Acknowledgments [io2d.general.ack]

1 The C++ programming language as described in this International Standard is based on the language as described in Chapter R (Reference Manual) of Stroustrup: *The C++ Programming Language* (second edition, Addison-Wesley Publishing Company, ISBN 0-201-53992-6, copyright ©1991 AT&T). That, in turn, is based on the C programming language as described in Appendix A of Kernighan and Ritchie: *The C Programming Language* (Prentice-Hall, 1978, ISBN 0-13-110163-3, copyright ©1978 AT&T).

2 Portions of the library Clauses of this International Standard are based on work by P.J. Plauger, which was published as *The Draft Standard C++ Library* (Prentice-Hall, ISBN 0-13-117003-1, copyright ©1995 P.J. Plauger).

3 POSIX® is a registered trademark of the Institute of Electrical and Electronic Engineers, Inc.

4 All rights in these originals are reserved.

1.6 Requirements [io2d.req]

1.6.1 Native handles [io2d.req.native]

1 Several classes described in this Technical Specification have members `native_handle_type` and `native_handle`. The presence of these members and their semantics is implementation-defined. [*Note:* These members allow implementations to provide access to implementation details. Their names are specified to facilitate portable compile-time detection. Actual use of these members is inherently non-portable. — *end note*]

2 Header <experimental/io2d> synopsis [io2d.syn]

```
#ifndef _IO2D_
#define _IO2D_

#include <memory>
#include <functional>
#include <exception>
#include <vector>
#include <string>
#include <algorithm>
#include <system_error>

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class io2d_error {
    success,
    no_memory,
    invalid_restore,
    invalid_pop_group,
    no_current_point,
    invalid_matrix,
    invalid_status,
    null_pointer,
    invalid_string,
    invalid_path_data,
    read_error,
    write_error,
    surface_finished,
    surface_type_mismatch,
    pattern_type_mismatch,
    invalid_content,
    invalid_format,
    invalid_visual,
    file_not_found,
    invalid_dash,
    invalid_dsc_comment,
    invalid_index,
    clip_not_representable,
    temp_file_error,
    invalid_stride,
    font_type_mismatch,
    user_font_immutable,
    user_font_error,
    negative_count,
    invalid_clusters,
    invalid_slant,
    invalid_weight,
    invalid_size,
    user_font_not_implemented,
```

```
  device_type_mismatch,
  device_error,
  invalid_mesh_construction,
  device_finished,
  last_value
};

enum class antialias {
  default_antialias,
  none,
  gray,
  subpixel,
  fast,
  good,
  best
};

enum class content {
  color,
  alpha,
  color_alpha
};

enum class fill_rule {
  winding,
  even_odd
};

enum class line_cap {
  butt,
  round,
  square
};

enum class line_join {
  miter,
  round,
  bevel,
  miter_or_bevel
};

enum class compositing_operator {
  clear,
  source,
  over,
  in,
  out,
  atop,
  dest,
  dest_over,
  dest_in,
  dest_out,
  dest_atop,
  xor_op,
  add,
```

```
    saturate,
    multiply,
    screen,
    overlay,
    darken,
    lighten,
    color_dodge,
    color_burn,
    hard_light,
    soft_light,
    difference,
    exclusion,
    hsl_hue,
    hsl_saturation,
    hsl_color,
    hsl_luminosity,
    default_op = over
  };

  enum class format {
    invalid,
    argb32,
    rgb24,
    a8,
    a1,
    rgb16_565,
    rgb30
  };

  enum class extend {
    none,
    repeat,
    reflect,
    pad,
    default_extend = none
  };

  enum class filter {
    fast,
    good,
    best,
    nearest,
    bilinear,
    gaussian,
    default_filter = good
  };

  enum class pattern_type {
    unknown,
    solid_color,
    linear,
    radial,
    mesh
  };
```

```
enum class font_slant {
  normal,
  italic,
  oblique
};

enum class font_weight {
  normal,
  bold
};

enum class subpixel_order {
  default_subpixel_order,
  horizontal_rgb,
  horizontal_bgr,
  vertical_rgb,
  vertical_bgr
};

enum class path_data_type {
  move_to,
  line_to,
  curve_to,
  new_sub_path,
  close_path,
  rel_move_to,
  rel_line_to,
  rel_curve_to,
  arc,
  arc_negative,
  change_matrix,
  change_origin
};

struct rectangle {
  double x;
  double y;
  double width;
  double height;
};

struct rgba_color {
  double r;
  double g;
  double b;
  double a;

  const static rgba_color alice_blue;
  const static rgba_color antique_white;
  const static rgba_color aqua;
  const static rgba_color aquamarine;
  const static rgba_color azure;
  const static rgba_color beige;
  const static rgba_color bisque;
  const static rgba_color black;
```

```
const static rgba_color blanched_almond;
const static rgba_color blue;
const static rgba_color blue_violet;
const static rgba_color brown;
const static rgba_color burly_wood;
const static rgba_color cadet_blue;
const static rgba_color chartreuse;
const static rgba_color chocolate;
const static rgba_color coral;
const static rgba_color cornflower_blue;
const static rgba_color cornsilk;
const static rgba_color crimson;
const static rgba_color cyan;
const static rgba_color dark_blue;
const static rgba_color dark_cyan;
const static rgba_color dark_goldenrod;
const static rgba_color dark_gray;
const static rgba_color dark_green;
const static rgba_color dark_grey;
const static rgba_color dark_khaki;
const static rgba_color dark_magenta;
const static rgba_color dark_olive_green;
const static rgba_color dark_orange;
const static rgba_color dark_orchid;
const static rgba_color dark_red;
const static rgba_color dark_salmon;
const static rgba_color dark_sea_green;
const static rgba_color dark_slate_blue;
const static rgba_color dark_slate_gray;
const static rgba_color dark_slate_grey;
const static rgba_color dark_turquoise;
const static rgba_color dark_violet;
const static rgba_color deep_pink;
const static rgba_color deep_sky_blue;
const static rgba_color dim_gray;
const static rgba_color dim_grey;
const static rgba_color dodger_blue;
const static rgba_color firebrick;
const static rgba_color floral_white;
const static rgba_color forest_green;
const static rgba_color fuchsia;
const static rgba_color gainsboro;
const static rgba_color ghost_white;
const static rgba_color gold;
const static rgba_color goldenrod;
const static rgba_color gray;
const static rgba_color green;
const static rgba_color green_yellow;
const static rgba_color grey;
const static rgba_color honeydew;
const static rgba_color hot_pink;
const static rgba_color indian_red;
const static rgba_color indigo;
const static rgba_color ivory;
const static rgba_color khaki;
```

```
const static rgba_color lavender;
const static rgba_color lavender_blush;
const static rgba_color lawn_green;
const static rgba_color lemon_chiffon;
const static rgba_color light_blue;
const static rgba_color light_coral;
const static rgba_color light_cyan;
const static rgba_color light_goldenrod_yellow;
const static rgba_color light_gray;
const static rgba_color light_green;
const static rgba_color light_grey;
const static rgba_color light_pink;
const static rgba_color light_salmon;
const static rgba_color light_sea_green;
const static rgba_color light_sky_blue;
const static rgba_color light_slate_gray;
const static rgba_color light_slate_grey;
const static rgba_color light_steel_blue;
const static rgba_color light_yellow;
const static rgba_color lime;
const static rgba_color lime_green;
const static rgba_color linen;
const static rgba_color magenta;
const static rgba_color maroon;
const static rgba_color medium_aquamarine;
const static rgba_color medium_blue;
const static rgba_color medium_orchid;
const static rgba_color medium_purple;
const static rgba_color medium_sea_green;
const static rgba_color medium_slate_blue;
const static rgba_color medium_spring_green;
const static rgba_color medium_turquoise;
const static rgba_color medium_violet_red;
const static rgba_color midnight_blue;
const static rgba_color mint_cream;
const static rgba_color misty_rose;
const static rgba_color moccasin;
const static rgba_color navajo_white;
const static rgba_color navy;
const static rgba_color old_lace;
const static rgba_color olive;
const static rgba_color olive_drab;
const static rgba_color orange;
const static rgba_color orange_red;
const static rgba_color orchid;
const static rgba_color pale_goldenrod;
const static rgba_color pale_green;
const static rgba_color pale_turquoise;
const static rgba_color pale_violet_red;
const static rgba_color papaya_whip;
const static rgba_color peach_puff;
const static rgba_color peru;
const static rgba_color pink;
const static rgba_color plum;
const static rgba_color powder_blue;
```

```
  const static rgba_color purple;
  const static rgba_color red;
  const static rgba_color rosy_brown;
  const static rgba_color royal_blue;
  const static rgba_color saddle_brown;
  const static rgba_color salmon;
  const static rgba_color sandy_brown;
  const static rgba_color sea_green;
  const static rgba_color sea_shell;
  const static rgba_color sienna;
  const static rgba_color silver;
  const static rgba_color sky_blue;
  const static rgba_color slate_blue;
  const static rgba_color slate_gray;
  const static rgba_color slate_grey;
  const static rgba_color snow;
  const static rgba_color spring_green;
  const static rgba_color steel_blue;
  const static rgba_color tan;
  const static rgba_color teal;
  const static rgba_color thistle;
  const static rgba_color tomato;
  const static rgba_color transparent_black; // Note: Not in CSS3.
  const static rgba_color turquoise;
  const static rgba_color violet;
  const static rgba_color wheat;
  const static rgba_color white;
  const static rgba_color white_smoke;
  const static rgba_color yellow;
  const static rgba_color yellow_green;
};

rgba_color operator*(const rgba_color& lhs, double rhs);
rgba_color& operator*=(rgba_color& lhs, double rhs);
bool operator==(const rgba_color& lhs, const rgba_color& rhs);
bool operator!=(const rgba_color& lhs, const rgba_color& rhs);

inline namespace literals {
  double operator""ubyte(unsigned long long value);
  double operator""unorm(long double value);
} // inline namespace literals

struct point {
  double x;
  double y;
};

point operator+(const point& lhs);
point operator+(const point& lhs, const point& rhs);
point operator+(const point& lhs, double rhs);
point& operator+=(point& lhs, const point& rhs);
point& operator+=(point& lhs, double rhs);
point operator-(const point& lhs);
point operator-(const point& lhs, const point& rhs);
point operator-(const point& lhs, double rhs);
```

```
  point& operator-=(point& lhs, const point& rhs);
  point& operator-=(point& lhs, double rhs);
  point operator*(const point& lhs, const point& rhs);
  point operator*(const point& lhs, double rhs);
  point& operator*=(point& lhs, const point& rhs);
  point& operator*=(point& lhs, double rhs);
  point operator/(const point& lhs, const point& rhs);
  point operator/(const point& lhs, double rhs);
  point& operator/=(point& lhs, const point& rhs);
  point& operator/=(point& lhs, double rhs);
  bool operator==(const point& lhs, const point& rhs);
  bool operator!=(const point& lhs, const point& rhs);

  struct glyph {
    unsigned long index;
    double x;
    double y;
  };

  struct text_cluster {
    int num_bytes;
    int num_glyphs;
  };

  struct font_extents {
    double ascent;
    double descent;
    double height;
    double max_x_advance;
    double max_y_advance;
  };

  struct text_extents {
    double x_bearing;
    double y_bearing;
    double width;
    double height;
    double x_advance;
    double y_advance;
  };

  struct matrix_2d {
    double m00;
    double m01;
    double m10;
    double m11;
    double m20;
    double m21;

    static matrix_2d init_identity();
    static matrix_2d init_translate(const point& value);
    static matrix_2d init_scale(const point& value);
    static matrix_2d init_rotate(double radians);
    static matrix_2d init_shear_x(double factor);
    static matrix_2d init_shear_y(double factor);
```

```
  matrix_2d& translate(const point& value);
  matrix_2d& scale(const point& value);
  matrix_2d& rotate(double radians);
  matrix_2d& shear_x(double factor);
  matrix_2d& shear_y(double factor);
  matrix_2d& invert();

  double determinant() const;
  point transform_distance(const point& dist) const;
  point transform_point(const point& pt) const;
};

matrix_2d operator*(const matrix_2d& lhs, const matrix_2d& rhs);
matrix_2d& operator*=(matrix_2d& lhs, const matrix_2d& rhs);
bool operator==(const matrix_2d& lhs, const matrix_2d& rhs);
bool operator!=(const matrix_2d& lhs, const matrix_2d& rhs);

struct path_data {
  path_data_type type;
  union {
      point move;
      point line;
      struct {
              point pt1;
              point pt2;
              point pt3;
      } curve;
      point origin;
      struct {
              point center;
              double radius;
              double angle1;
              double angle2;
      } arc;
      matrix_2d matrix;
      int unused;
  } data;
};

bool operator==(const path_data& lhs, const path_data& rhs);
bool operator!=(const path_data& lhs, const path_data& rhs);

class io2d_error_category : public ::std::error_category {
public:
  virtual const char* name() const noexcept override;
  virtual ::std::string message(int errVal) const override;
  virtual bool equivalent(const ::std::error_code& ec, int condition) const
  noexcept override;
};

const ::std::error_category& io2d_category() noexcept;

// Forward declaration.
class path_factory;
```

```
class path {
public:
  typedef implementation-defined native_handle_type;

  path() = delete;
  path(const path_factory& pb);
  path(const path& other) = default;
  path& operator=(const path& other) = default;
  path(path&& other);
  path& operator=(path&& other);

  ::std::vector<path_data> get_data() const;
  const ::std::vector<path_data>& get_data_ref() const;
  rectangle get_path_extents() const;
};

class path_factory {
public:
  path_factory();
  path_factory(const path_factory& other);
  path_factory& operator=(const path_factory& other);
  path_factory(path_factory&& other);
  path_factory& operator=(path_factory&& other);

  path get_path() const;
  rectangle get_path_extents() const;

  void append(const path& p);
  void append(const path_factory& p);
  void append(const ::std::vector<path_data>& p);
  bool has_current_point() const;
  point get_current_point() const;
  void new_sub_path();
  void close_path();
  void arc(const point& center, double radius, double angle1, double angle2);
  void arc_negative(const point& center, double radius, double angle1, double
  angle2);
  void curve_to(const point& pt0, const point& pt1, const point& pt2);
  void line_to(const point& pt);
  void move_to(const point& pt);
  void rect(const rectangle& r);
  void rel_curve_to(const point& dpt0, const point& dpt1, const point& dpt2);
  void rel_line_to(const point& dpt);
  void rel_move_to(const point& dpt);

  void set_transform_matrix(const matrix_2d& m);
  matrix_2d get_transform_matrix() const;
  void set_origin(const point& pt);
  point get_origin() const;

  ::std::vector<path_data> get_data() const;
  path_data get_data(unsigned int index) const;
  const ::std::vector<path_data>& get_data_ref() const;
```

```
  void reset();
};

class device {
public:
  typedef implementation-defined native_handle_type;
  native_handle_type native_handle() const;

  device() = delete;
  device(const device&) = delete;
  device& operator=(const device&) = delete;
  device(device&& other);
  device& operator=(device&& other);
  explicit device(native_handle_type nh);
  void flush();
  void lock();
  void unlock();
};

// Forward declaration.
class font_options;

class font_options_factory {
public:
  font_options_factory();
  font_options_factory(const font_options_factory&);
  font_options_factory& operator=(const font_options_factory&);
  font_options_factory(font_options_factory&& other);
  font_options_factory& operator=(font_options_factory&& other);

  font_options get_font_options() const;
  void set_antialias(antialias a);
  antialias get_antialias() const;
  void set_subpixel_order(subpixel_order so);
  subpixel_order get_subpixel_order() const;
};

class font_options {
public:
  typedef implementation-defined native_handle_type;
  native_handle_type native_handle() const;

  font_options() = delete;
  font_options(const font_options&) = default;
  font_options& operator=(const font_options&) = default;
  font_options(font_options&& other);
  font_options& operator=(font_options&& other);
  font_options(antialias a, subpixel_order so);
  explicit font_options(native_handle_type nh);

  antialias get_antialias() const;
  subpixel_order get_subpixel_order() const;
};

class font_face {
```

```
public:
  typedef implementation-defined native_handle_type;
  native_handle_type native_handle() const;

  font_face() = delete;
  font_face(const font_face&) = default;
  font_face& operator=(const font_face&) = default;
  font_face(font_face&& other);
  font_face& operator=(font_face&& other);

  explicit font_face(native_handle_type nh);

  virtual ~font_face();
};

class scaled_font {
public:
  typedef implementation-defined native_handle_type;
  native_handle_type native_handle() const;

  scaled_font() = delete;
  scaled_font(const scaled_font&) = default;
  scaled_font& operator=(const scaled_font&) = default;
  scaled_font(scaled_font&& other);
  scaled_font& operator=(scaled_font&& other);

  explicit scaled_font(native_handle_type nh);
  scaled_font(const font_face& ff, const matrix_2d& fm, const matrix_2d& ctm,
  const font_options& fo);

  font_extents get_extents() const;
  text_extents get_text_extents(const ::std::string& utf8) const;
  text_extents get_glyph_extents(const ::std::vector<glyph>& glyphs) const;
  ::std::vector<glyph> text_to_glyphs(double x, double y, const
  ::std::string& utf8) const;
  ::std::vector<glyph> text_to_glyphs(double x, double y, const
  ::std::string& utf8, ::std::vector<text_cluster>& clusters, bool&
  clustersToGlyphsReverseMap) const;
};

class simple_font_face : public font_face {
public:
  simple_font_face() = delete;
  simple_font_face(const simple_font_face&) = default;
  simple_font_face& operator=(const simple_font_face&) = default;
  simple_font_face(const ::std::string& family, font_slant slant, font_weight
  weight);
  simple_font_face(simple_font_face&& other);
  simple_font_face& operator=(simple_font_face&& other);

  ::std::string get_family() const;
  font_slant get_slant() const;
  font_weight get_weight() const;
};
```

```
  // Forward declaration.
  class image_surface;

  // Forward declaration.
  class linear_pattern_factory;
  class mesh_pattern_factory;
  class radial_pattern_factory;
  class raster_source_pattern_factory;
  class solid_color_pattern_factory;
  class surface;

  class pattern {
  public:
    typedef implementation-defined native_handle_type;
    native_handle_type native_handle() const;

    pattern() = delete;
    pattern(const pattern&) = default;
    pattern& operator=(const pattern&) = default;
    pattern(pattern&& other);
    pattern& operator=(pattern&& other);

    ~pattern();

    pattern_type get_type() const;
  };

  class solid_color_pattern_factory {
  public:
    solid_color_pattern_factory();
    solid_color_pattern_factory(const solid_color_pattern_factory&);
    solid_color_pattern_factory& operator=(const solid_color_pattern_factory&);
    solid_color_pattern_factory(solid_color_pattern_factory&& other);
    solid_color_pattern_factory& operator=(solid_color_pattern_factory&& other);
    solid_color_pattern_factory(const rgba_color& color);

    pattern get_pattern() const;
    void set_extend(extend e);
    extend get_extend() const;
    void set_filter(filter f);
    filter get_filter() const;
    void set_matrix(const matrix_2d& m);
    matrix_2d get_matrix() const;

    rgba_color get_rgba() const;
    void set_rgba(const rgba_color& color);
    double get_red() const;
    void set_red(double red);
    double get_green() const;
    void set_green(double green);
    double get_blue() const;
    void set_blue(double blue);
    double get_alpha() const;
    void set_alpha(double alpha);
  };
```

```
class linear_pattern_factory {
public:
  linear_pattern_factory();
  linear_pattern_factory(const linear_pattern_factory&);
  linear_pattern_factory& operator=(const linear_pattern_factory&);
  linear_pattern_factory(linear_pattern_factory&& other);
  linear_pattern_factory& operator=(linear_pattern_factory&& other);
  linear_pattern_factory(const point& pt0, const point& pt1);

  pattern get_pattern() const;
  void set_extend(extend extend);
  extend get_extend() const;
  void set_filter(filter filter);
  filter get_filter() const;
  void set_matrix(const matrix_2d& matrix);
  matrix_2d get_matrix() const;

  void add_color_stop_rgba(double offset, const rgba_color& color);
  int get_color_stop_count() const;

  void get_color_stop_rgba(unsigned int index, double& offset, rgba_color&
  color) const;
  void set_color_stop_rgba(unsigned int index, double offset, const
  rgba_color& color);
  void get_linear_points(point& pt0, point& pt1) const;
  void set_linear_points(const point& pt0, const point& pt1);
};

class radial_pattern_factory {
public:
  radial_pattern_factory();
  radial_pattern_factory(const radial_pattern_factory&);
  radial_pattern_factory& operator=(const radial_pattern_factory&);
  radial_pattern_factory(radial_pattern_factory&& other);
  radial_pattern_factory& operator=(radial_pattern_factory&& other);
  radial_pattern_factory(const point& center0, double radius0, const point&
  center1, double radius1);

  pattern get_pattern() const;
  void set_extend(extend extend);
  extend get_extend() const;
  void set_filter(filter filter);
  filter get_filter() const;
  void set_matrix(const matrix_2d& matrix);
  matrix_2d get_matrix() const;

  void add_color_stop_rgba(double offset, const rgba_color& color);
  int get_color_stop_count() const;

  void get_color_stop_rgba(unsigned int index, double& offset, rgba_color&
  color) const;
  void set_color_stop_rgba(unsigned int index, double offset, const
  rgba_color& color);
```

```
  void get_radial_circles(point& center0, double& radius0, point& center1,
  double& radius1) const;
  void set_radial_circles(const point& center0, double radius0, const point&
  center1, double radius1);
};

class mesh_pattern_factory {
public:
  mesh_pattern_factory();
  mesh_pattern_factory(const mesh_pattern_factory&);
  mesh_pattern_factory& operator=(const mesh_pattern_factory&);
  mesh_pattern_factory(mesh_pattern_factory&& other);
  mesh_pattern_factory& operator=(mesh_pattern_factory&& other);

  pattern get_pattern() const;
  void set_extend(extend extend);
  extend get_extend() const;
  void set_filter(filter filter);
  filter get_filter() const;
  void set_matrix(const matrix_2d& matrix);
  matrix_2d get_matrix() const;

  void begin_patch();
  void begin_edit_patch(unsigned int patch_num);
  void end_patch();
  void move_to(const point& pt);
  void line_to(const point& pt);
  void curve_to(const point& pt0, const point& pt1, const point& pt2);
  void set_control_point(unsigned int point_num, const point& pt);
  void set_corner_color_rgba(unsigned int corner_num, const rgba_color&
  color);
  unsigned int get_patch_count() const;
  path get_path(unsigned int patch_num) const;
  path_factory get_path_factory(unsigned int patch_num) const;
  point get_control_point(unsigned int patch_num, unsigned int point_num)
  const;
  rgba_color get_corner_color_rgba(unsigned int patch_num, unsigned int
  corner_num) const;
};

// 29
class surface {
public:
  // See 1.6.1
  typedef implementation-defined native_handle_type;
  native_handle_type native_handle() const;

  // tuple<dashes, offset>
  typedef ::std::tuple<::std::vector<double>, double> dashes;

  // 29.3, constructors, assignment operators, destructors:
  surface() = delete;
  surface(const surface&) = delete;
  surface& operator=(const surface&) = delete;
  surface(surface&& other);
```

```
surface& operator=(surface&& other);
surface(const surface& other, content content, int width, int height);
virtual ~surface();

// 29.4, state modifiers:
void finish();
void flush();
::std::shared_ptr<device> get_device();
void mark_dirty();
void mark_dirty(const rectangle& rect);
void set_device_offset(const point& offset);
void write_to_png(const ::std::string& filename);
image_surface map_to_image();
image_surface map_to_image(const rectangle& extents);
void unmap_image(image_surface& image);
void save();
void restore();
void set_pattern();
void set_pattern(const pattern& source);
void set_antialias(antialias a);
void set_dashes();
void set_dashes(const dashes& d);
void set_fill_rule(fill_rule fr);
void set_line_cap(line_cap lc);
void set_line_join(line_join lj);
void set_line_width(double width);
void set_miter_limit(double limit);
void set_compositing_operator(compositing_operator co);
void clip();
void reset_clip();
void set_path();
void set_path(const path& p);

// 29.5, render modifiers:
void fill();
void fill(const surface& s);
void mask(const pattern& pttn);
void mask(const surface& surface);
void mask(const surface& surface, const point& origin);
void paint();
void paint(const surface& s);
void paint(double alpha);
void paint(const surface& s, double alpha);
void stroke();
void stroke(const surface& s);
void show_text(const ::std::string& utf8);
void show_glyphs(const ::std::vector<glyph>& glyphs);
void show_text_glyphs(const ::std::string& utf8,
  const ::std::vector<glyph>& glyphs,
  const ::std::vector<text_cluster>& clusters,
  bool clusterToGlyphsMapReverse = false);

// 29.6, transformation modifiers:
void set_matrix(const matrix_2d& matrix);
```

```
    // 29.7, font modifiers:
    void select_font_face(const ::std::string& family, font_slant slant,
      font_weight weight);
    void set_font_size(double size);
    void set_font_matrix(const matrix_2d& matrix);
    void set_font_options(const font_options& options);
    void set_font_face(const font_face& font_face);
    void set_scaled_font(const scaled_font& scaled_font);

    // 29.8, state observers:
    content get_content() const;
    point get_device_offset() const;
    bool has_surface_resource() const;
    pattern get_pattern() const;
    antialias get_antialias() const;
    int get_dashes_count() const;
    dashes get_dashes() const;
    fill_rule get_fill_rule() const;
    line_cap get_line_cap() const;
    line_join get_line_join() const;
    double get_line_width() const;
    double get_miter_limit() const;
    compositing_operator get_compositing_operator() const;
    double get_tolerance() const;
    rectangle get_clip_extents() const;
    bool in_clip(const point& pt) const;
    ::std::vector<rectangle> get_clip_rectangles() const;

    // 29.9, render observers:
    rectangle get_fill_extents() const;
    bool in_fill(const point& pt) const;
    rectangle get_stroke_extents() const;
    bool in_stroke(const point& pt) const;
    font_extents get_font_extents() const;
    text_extents get_text_extents(const ::std::string& utf8) const;
    text_extents get_glyph_extents(const ::std::vector<glyph>& glyphs) const;

    // 29.10, transformation observers:
    matrix_2d get_matrix() const;
    point user_to_device() const;
    point user_to_device_distance() const;
    point device_to_user() const;
    point device_to_user_distance() const;

    // 29.11, font observers:
    matrix_2d get_font_matrix() const;
    font_options get_font_options() const;
    font_face get_font_face() const;
    scaled_font get_scaled_font() const;
  };

  class image_surface : public surface {
  public:
    image_surface() = delete;
    image_surface(const image_surface&) = delete;
```

```
    image_surface& operator=(const image_surface&) = delete;
    image_surface(image_surface&& other);
    image_surface& operator=(image_surface&& other);
    image_surface(surface::native_handle_type nh, surface::native_handle_type
    map_of);
    image_surface(format format, int width, int height);
    image_surface(vector<unsigned char>& data, format format, int width, int
    height, int stride);
    // create_similar_image
    image_surface(const surface& other, format format, int width, int height);
    // create_from_png
    image_surface(const ::std::string& filename);

    void set_data(::std::vector<unsigned char>& data);
    ::std::vector<unsigned char> get_data() const;
    format get_format() const;
    int get_width() const;
    int get_height() const;
    int get_stride() const;
  };

  int format_stride_for_width(format format, int width);
  surface make_surface(implementation-defined) implementation-defined;
  image_surface make_image_surface(format format, int width, int height);
} } } } // namespaces std::experimental::io2d::v1

namespace std {
        template<>
        struct is_error_condition_enum<::std::experimental::io2d::io2d_error>
                : public ::std::true_type{ };

        template<>
        struct is_error_code_enum<::cairo_status_t>
                : public ::std::true_type{ };

        ::std::error_condition make_error_condition(experimental::io2d::io2d_error
        e) noexcept;

        ::std::error_code make_error_code(cairo_status_t e) noexcept;
} // namespace std

#endif
```

3 Enum class `io2d_error` [io2d.io2d.error]

3.1 `io2d_error` Summary [io2d.io2d.error.summary]

1 The `io2d_error` enum class is used as an `error_condition` associated with `io2d_error_category`. See Table 1 for the meaning of each status code.

3.2 `io2d_error` Synopsis [io2d.io2d.error.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class io2d_error {
    success,
    no_memory,
    invalid_restore,
    invalid_pop_group,
    no_current_point,
    invalid_matrix,
    invalid_status,
    null_pointer,
    invalid_string,
    invalid_path_data,
    read_error,
    write_error,
    surface_finished,
    surface_type_mismatch,
    pattern_type_mismatch,
    invalid_content,
    invalid_format,
    invalid_visual,
    file_not_found,
    invalid_dash,
    invalid_dsc_comment,
    invalid_index,
    clip_not_representable,
    temp_file_error,
    invalid_stride,
    font_type_mismatch,
    user_font_immutable,
    user_font_error,
    negative_count,
    invalid_clusters,
    invalid_slant,
    invalid_weight,
    invalid_size,
    user_font_not_implemented,
    device_type_mismatch,
    device_error,
    invalid_mesh_construction,
    device_finished,
    last_status
  };
} } } } // namespaces std::experimental::io2d::v1
```

3.3 `io2d_error` Enumerators [io2d.io2d.error.enumerators]

Table 1 — `io2d_error` enumerator meanings

Enumerator	Meaning
`success`	The operation completed successfully.
`no_memory`	The operation failed due to insufficient memory. [*Note:* Deprecated. Implementations should throw an exception of type `std::bad_alloc` instead. — *end note*]
`invalid_restore`	A call was made to `surface::restore` for which no prior call to `surface::save` was made.
`invalid_pop_group`	A call was made to `surface::pop_group` or `surface::pop_group_to_source` for which no prior call to `surface::push_group` was made.
`no_current_point`	The operation requires a current point but no current point was set. This is usually the result of a call to `path_builder::rel_curve_to`, `path_builder::rel_line_to`, or `path_builder::rel_move_to` and can be corrected by first establishing a current point with a non-"rel" member function call such as `path_builder::move_to`.
`invalid_matrix`	A `matrix_2d` that the operation depends on is invalid. To be valid a `matrix_2d` must be invertible.
`invalid_status`	An internal error has occurred. [*Note:* This value must only be used when no other `io2d_error` value is appropriate. It signifies that an implementation-specific error occurred such as passing a bad native handle as an argument. The conditions and circumstances under which this `io2d_error` value occurs are implementation-defined. — *end note*]
`null_pointer`	A null pointer value was unexpectedly encountered. [*Note:* The conditions and circumstances under which this `io2d_error` value occurs are implementation-defined. — *end note*]
`invalid_string`	A UTF-8 string value was expected but the string is not a valid UTF-8 string.
`invalid_path_data`	Invalid data was encountered in a `path` or a `path_builder` object. [*Note:* This status value should only occur when a user directly manipulates the internal data of a `path` or a `path_builder` object. Creating a `path` object using the `path_builder` type should never result in an invalid path unless the user calls the non-`const` overload of `path_builder::get_data_ref` and directly manipulates the data in an erroneous manner. — *end note*]
`read_error`	An error occurred while attempting to read data from an input stream.
`write_error`	An error occurred while attempting to write data to an output stream.

Table 1 — `io2d_error` enumerator meanings (continued)

Enumerator	Meaning
`surface_finished`	An attempt was made to use or manipulate a `surface` object or `surface`-derived object which is no longer valid. [*Note:* This can occur due to a previous call to `surface::finish` or as a result of erroneous usage of a native handle. — *end note*]
`surface_type_mismatch`	An operation was attempted on a surface that does not support it. [*Note:* Deprecated. — *end note*]
`pattern_type_mismatch`	An operation was attempted on a pattern that does not support it. [*Note:* Deprecated. — *end note*]
`invalid_content`	An invalid `content` value was passed as an argument. [*Note:* Deprecated. Implementations should throw an exception of type `std::invalid_argument` instead. — *end note*]
`invalid_format`	An invalid `format` value was passed as an argument. [*Note:* Deprecated. Implementations should throw an exception of type `std::invalid_argument` instead. — *end note*]
`invalid_visual`	Invalid visual. [*Note:* Deprecated. Implementations should use `invalid_status` instead. — *end note*]
`file_not_found`	File not found.
`invalid_dash`	An invalid dash value was specified in a call to `surface::set_dashes`.
`invalid_dsc_comment`	Invalid dsc comment. [*Note:* Deprecated. Implementations should use `invalid_status` instead. — *end note*]
`invalid_index`	Invalid index. [*Note:* Deprecated. Implementations should throw an exception of type `std::out_of_range` instead. — *end note*]
`clip_not_representable`	A call was made to `surface::get_clip_rectangles` when the `surface` object's current clipping region could not be represented with rectangles.
`temp_file_error`	Temp file error. [*Note:* Deprecated. Implementations that require temporary files should use `invalid_status` instead. — *end note*]
`invalid_stride`	An invalid stride value was used. Surface formats may require padding at the end of each row of pixel data depending on the implementation and the current graphics chipset, if any. Use `format_stride_for_width` to obtain the correct stride value.
`font_type_mismatch`	Font type mismatch. [*Note:* Deprecated. — *end note*]
`user_font_immutable`	User font immutable. [*Note:* Reserved. — *end note*]
`user_font_error`	User font error. [*Note:* Reserved. — *end note*]
`negative_count`	Negative count. [*Note:* Deprecated. — *end note*]
`invalid_clusters`	A call was made to `surface::show_text_glyphs` with a `std::vector<text_clusters>` argument that does not properly map the UTF-8 `std::string` code points to the `std::vector<glyph>` glyphs.

Table 1 — `io2d_error` enumerator meanings (continued)

Enumerator	Meaning
`invalid_slant`	Invalid slant. [*Note:* Deprecated. — *end note*]
`invalid_weight`	Invalid weight. [*Note:* Deprecated. — *end note*]
`invalid_size`	The input value for a size parameter is incorrect. [*Note:* Typically this is an improper width or height argument. It could be zero or it could be a value that exceeds the implementation-defined size limits for a surface or a pattern. — *end note*]
`user_font_not_implemented`	User fonts are not implemented. [*Note:* Reserved. — *end note*]
`device_type_mismatch`	Device type mismatch. [*Note:* Deprecated. — *end note*]
`device_error`	The operation failed. The `device` encountered an error. [*Note:* The conditions and circumstances in which this `io2d_error` value occurs are implementation-defined. — *end note*]
`invalid_mesh_construction`	A mesh construction operation on a `mesh_pattern_builder` object failed. Mesh construction operations are only permitted in between a call to either `mesh_pattern_builder::begin_patch` or `mesh_pattern_builder::begin_edit_patch` and `mesh_pattern_builder::end_patch`.
`device_finished`	Device finished. [*Note:* Deprecated. — *end note*]
`last_status`	This value is unused. [*Note:* Deprecated. — *end note*]

4 Enum class `antialias` [io2d.antialias]

4.1 `antialias` Summary [io2d.antialias.summary]

1 The antialias enum class specifies the type of antialiasing that the rendering system is requested to use for rendering `surface` and `pattern` objects, and for rendering text. See Table 2 for the meaning of each `antialias` enumerator.

2 [*Note:* The value `antialias::subpixel` is only expected to be meaningful in the context of rendering text. — *end note*]

4.2 `antialias` Synopsis [io2d.antialias.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class antialias {
    default_antialias,
    none,
    gray,
    subpixel,
    fast,
    good,
    best
  };
} } } } // namespaces std::experimental::io2d::v1
```

4.3 `antialias` Enumerators [io2d.antialias.enumerators]

Table 2 — `antialias` enumerator meanings

Enumerator	Meaning
`default_antialias`	implementation-defined.
`none`	No antialiasing.
`gray`	Monochromatic antialiasing.
`subpixel`	Antialiasing that breaks pixels into their constituent color channels and manipulates those color channels individually. The meaning of this value for any rendering operation other than `surface::show_text`, `surface::show_glyphs`, and `surface::show_text_glyphs` is implementation-defined.
`fast`	implementation-defined. [*Note:* By choosing this value, the user is hinting that some antialiasing is desired but that performance is more important that appearance. — *end note*]
`good`	implementation-defined. [*Note:* By choosing this value, the user is hinting that antialiasing is desired and that sacrificing some performance to obtain antialiasing is acceptable. Implementations that provide antialiasing shall enable some form of antialiasing when this option is selected unless there is a compelling performance reason not to do so.

Table 2 — `antialias` enumerator meanings (continued)

Enumerator	Meaning
`best`	implementation-defined. [*Note:* By choosing this value, the user is hinting that antialiasing is more important than performance. Implementations that provide antialiasing shall enable some form of antialiasing when this option is selected, preferably the form that the implementor believes to be the form that generally provides the best visual results.

5 Enum class `content` [io2d.content]

5.1 `content` Summary [io2d.content.summary]

1 The `content` enum class describes the type of data that a `surface` object contains. See Table 3 for the meaning of each enumerator.

5.2 `content` Synopsis [io2d.content.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class content {
    color,
    alpha,
    color_alpha
  };
} } } } // namespaces std::experimental::io2d::v1
```

5.3 `content` Enumerators [io2d.content.enumerators]

Table 3 — `content` value meanings

Enumerator	Meaning
`color`	The `surface` holds opaque color data only.
`alpha`	The `surface` holds alpha (translucency) data only.
`color_alpha`	The `surface` holds both color and alpha data.

6 Enum class `fill_rule` [io2d.fill.rule]

6.1 `fill_rule` Summary [io2d.fill.rule.summary]

1 The `fill_rule` enum class determines how the individuals paths in `path` objects are filled. For information about multiple paths in a single `path` object, see the description of the `path` class (23).

2 In order to determine whether a point will be included in a fill operation, create a ray from the point to infinity. The direction of the ray does not matter provided that it does not pass through the end point of a path segment and does not intersect the path at a point that is tangent to the path. The intersections of the ray with the path are used in conjunction with the current `fill_rule` value to determine whether a point is filled. See Table 4 for the meaning of each `fill_rule` enumerator. [*Note:* When used below, if the term path is formatted normally, i.e. path, it refers to an individual path within a `path` object. If the term is formatted in code style, i.e. `path`, it refers to the `path` type, which may contain one or more paths. — *end note*]

3 The default `fill_rule` is `fill_rule::winding`.

6.2 `fill_rule` Synopsis [io2d.fill.rule.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class fill_rule {
    winding,
    even_odd
  };
} } } } // namespaces std::experimental::io2d::v1
```

6.3 `fill_rule` Enumerators [io2d.fill.rule.enumerators]

Table 4 — `fill_rule` enumerator meanings

Enumerator	Meaning
`winding`	Starting with a count of zero, whenever the path crosses the ray from left to right, add one to the count, and whenever the path crosses the ray from right to left, subtract one from the count. If the count is zero the point is not filled, otherwise it is filled.
`even_odd`	Count the number of times the ray intersects the path. If the count is even, the point is not filled, otherwise it is filled.

7 Enum class `line_cap` [io2d.line.cap]

7.1 `line_cap` Summary [io2d.line.cap.summary]

1 The `line_cap` enum class specifies how the ends of lines should be rendered when a `path` is stroked. See Table 5 for the meaning of each `line_cap` enumerator.

7.2 `line_cap` Synopsis [io2d.line.cap.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class line_cap {
    butt,
    round,
    square
  };
} } } } // namespaces std::experimental::io2d::v1
```

7.3 `line_cap` Enumerators [io2d.line.cap.enumerators]

Table 5 — `line_cap` enumerator meanings

Enumerator	Meaning
`butt`	The line has no cap. It terminates exactly at the end `point`.
`round`	The line has a circular cap, with the end `point` serving as the center of the circle and the line width serving as its diameter.
`square`	The line has a square cap, with the end `point` serving as the center of the square and the line width serving as the length of each side.

8 Enum class `line_join` [io2d.line.join]

8.1 `line_join` Summary [io2d.line.join.summary]

1 The `line_join` enum class specifies how the junction of two line segments should be rendered when a `path` is stroked. See Table 6 for the meaning of each enumerator.

8.2 `line_join` Synopsis [io2d.line.join.synopsis]

```
namespace std { namespace experimental { namespace drawing { inline namespace
v1 {
  enum class line_join {
    miter,
    round,
    bevel,
    miter_or_bevel
  };
} } } } // namespaces std::experimental::graphics::v1
```

8.3 `line_join` Enumerators [io2d.line.join.enumerators]

Table 6 — `line_join` enumerator meanings

Enumerator	Meaning
`miter`	Joins will be angular. [*Note:* When the join angle is less than 0.0005 radians, it is implementation-defined whether this value will produce mitered or beveled joins. Specifically, when this value is set implementations may switch to beveled rendering instead of mitered rendering at any arbitrary *implementation-defined*angle value, provided that the value is less than 0.0005 radians and that all joins with angles below the *implementation-defined*angle value are rendered as beveled. — *end note*]
`round`	Joins will be rounded, with the center of the circle being the join point.
`bevel`	Joins will be beveled, with the join cut off at half the line width from the join point. [*Note:* Implementations may vary the cut off distance by an amount that is less than one pixel at each join for aesthetic or technical reasons. — *end note*]
`miter_or_bevel`	Joins will be angular or beveled, depending on the current miter limit.

9 Enum class `compositing_operator` [io2d.compositing.operator]

9.1 `compositing_operator` Summary [io2d.compositing.operator.summary]

1 The `compositing_operator` enum class determines how drawing operations process the source (input) content and the existing destination content to compose the final result content that will replace the original destination content. See Table 7, Table 8 and Table 9 for the meaning of each `compositing_operator` enumerator.

9.2 `compositing_operator` Synopsis [io2d.compositing.operator.synopsis]

```
namespace std { namespace experimental { namespace drawing { inline namespace
v1 {
  enum class compositing_operator {
    over,
    clear,
    source,
    in,
    out,
    atop,
    dest,
    dest_over,
    dest_in,
    dest_out,
    dest_atop,
    xor_op,
    add,
    saturate,
    multiply,
    screen,
    overlay,
    darken,
    lighten,
    color_dodge,
    color_burn,
    hard_light,
    soft_light,
    difference,
    exclusion,
    hsl_hue,
    hsl_saturation,
    hsl_color,
    hsl_luminosity
  };
} } } } // namespaces std::experimental::graphics::v1
```

9.3 `compositing_operator` Enumerators [io2d.compositing.operator.enumerators]

1 The tables below specify the compositing operations by mathematical formula. The formulas differentiate between three color channels (red, green, and blue) and a transparency channel (alpha). Channel values are

in the range $[0.0, 1.0]$.

2 The following symbols and specifiers are used:
The R symbol means the result color value
The S symbol means the source color value
The D symbol means the destination color value
The c specifier means the color channels of the value it follows
The a specifier means the alpha channel of the value it follows

3 The color symbols R, S, and D may appear with or without any specifiers.

4 If a color symbol appears alone, it designates the entire color as a tuple in the unsigned normalized form (red, green, blue, alpha).

5 The specifiers c and a may appear alone or together after any of the three color symbols.

6 The presence of the c specifier alone means the three color channels of the color as a tuple in the unsigned normalized form (red, green, blue).

7 The presence of the a specifier alone means the alpha channel of the color in unsigned normalized form.

8 The presence of the specifiers together in the form ca means the value of the color as a tuple in the unsigned normalized form (red, green, blue, alpha), where the value of each color channel is the product of each color channel and the alpha channel and the value of the alpha channel is the original value of the alpha channel. [*Example:* When it appears in a formula, Sca means $((Sc \times Sa), Sa)$, such that, given a source color $Sc = (1.0, 0.5, 0.0)$ and an source alpha $Sa = (0.5)$, the value of Sca when specified in one of the formulas would be $Sca = (1.0 \times 0.5, 0.5 \times 0.5, 0.0 \times 0.5, 0.5) = (0.5, 0.25, 0.0, 0.5)$. The same is true for Dca and Rca. — *end example*]

9 No space is left between a value and its channel specifiers. Channel specifiers will be preceded by exactly one value symbol.

10 When performing an operation that involves evaluating the color channels, each color channel should be evaluated individually to produce its own value.

11 The basic enumerators specify a value for Bounding. This value may be Bound, Unbound, or N/A.

12 If the Bounding value is 'Bound", then the source is treated as though it is also a mask. As such, only areas of the surface where the source would affect the surface are altered. The remaining areas of the surface have the same color value as before the compositing operation. [*Note:* If a clip is set then it will constrain the area of the surface that the compositing operation will affect to be the intersection of the source and the clip. If they do not intersect then no change will occur. — *end note*]

13 If the Bounding value is 'Unbound', then every area of the surface that is not affected by the source will become transparent black. In effect, it is as though the source was treated as being the same size as the destination surface with every part of the source that does not already have a color value assigned to it being treated as though it were transparent black. Application of the formula with this precondition results in those areas evaluating to transparent black such that evaluation can be bypassed due to the predetermined outcome. [*Note:* If a clip is set then only those areas of the surface that the are within the clip will be affected by the compositing operation. Even if no part of the source is within the clip, the operation will still set every area within the clip to transparent black. Areas outside the clip are not modified. — *end note*]

14 If the Bounding value is 'N/A', the operation would have the same effect regardless of whether it was treated as Bound or Unbound such that those Bounding values are not applicable to the operation. An N/A formula when applied to an area where the source does not provide a value will evaluate to the original value of the destination even if the source is treated as having a value there of transparent black. As such the result is the same as if the source were treated as being a mask, i.e. Bound and Unbound treatment each produce the same result in areas where the source does not have a value. [*Note:* If a clip is set then only those areas

of the surface that the are within the clip will be affected by the compositing operation. — *end note*]

Table 7 — `compositing_operator` basic enumerator meanings

Enumerator	Bounding	Color	Alpha
`clear`	Bound	$Rc = 0$	$Ra = 0$
`source`	Bound	$Rc = Sc$	$Ra = Sa$
`over`	N/A	$Rc = \frac{(Sca + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da \times (1 - Sa)$
`in`	Unbound	$Rc = Sc$	$Ra = Sa \times Da$
`out`	Unbound	$Rc = Sc$	$Ra = Sa \times (1 - Da)$
`atop`	N/A	$Rc = Sca + Dc \times (1 - Sa)$	$Ra = Da$
`dest`	N/A	$Rc = Dc$	$Ra = Da$
`dest_over`	N/A	$\frac{(Sca \times (1 - Da) + Dca)}{Ra}$	$(1 - Da) \times Sa + Da$
`dest_in`	Unbound	$Rc = Dc$	$Ra = Sa \times Da$
`dest_out`	N/A	$Rc = Dc$	$Ra = (1 - Sa) \times Da$
`dest_atop`	Unbound	$Rc = Sc \times (1 - Da) + Dca$	$Ra = Sa$
`xor_op`	N/A	$Rc = \frac{(Sca \times (1 - Da) + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da - 2 \times Sa \times Da$
`add`	N/A	$Rc = \frac{(Sca + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$
`saturate`	N/A	$Rc = \frac{(min(Sa, 1 - Da) \times Sc + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$

15 The blend enumerators and hsl enumerators share a common formula for the result color's color channel, with only one part of it changing depending on the enumerator. The result color's color channel value formula is as follows: $Rc = \frac{1}{Ra} \times ((1 - Da) \times Sca + (1 - Sa) \times Dca + Sa \times Da \times f(Sc, Dc))$. The function $f(Sc, Dc)$ is the component of the formula that is enumerator dependent.

16 For the blend enumerators, the color channels shall be treated as separable, meaning that the color formula shall be evaluated separately for each color channel: red, green, and blue.

17 The color formula divides 1 by the result color's alpha channel value. As a result, if the result color's alpha channel is zero then a division by zero would normally occur. Implementations shall not throw an exception nor otherwise produce any observable error condition if the result color's alpha channel is zero. Instead, implementations shall bypass the division by zero and produce the result color (0.0, 0.0, 0.0, 0.0), i.e. *transparent black*, if the result color alpha channel formula evaluates to zero. [*Note:* The simplest way to comply with this requirement is to bypass evaluation of the color channel formula in the event that the result alpha is zero. However, in order to allow implementations the greatest latitude possible, only the result is specified. — *end note*]

18 For the enumerators in Table 8 and Table 9 the result color's alpha channel value formula is as follows: $Ra = Sa + Da \times (1 - Sa)$. [*Note:* Since it is the same formula for all enumerators in those tables, the formula is not included in those tables. — *end note*]

19 All of the blend enumerators and hsl enumerators have a Bounding value of N/A.

Table 8 — `compositing_operator` blend enumerator meanings

Enumerator	Color
`multiply`	$f(Sc, Dc) = Sc \times Dc$

Table 8 — `compositing_operator` blend enumerator meanings (continued)

Enumerator	**Color**
`screen`	$f(Sc, Dc) = Sc + Dc - Sc \times Dc$

`overlay`	$if(Dc \leq 0.5)$ {
	$f(Sc, Dc) = 2 \times Sc \times Dc$
	}
	else {
	$f(Sc, Dc) =$
	$1 - 2 \times (1 - Sc) \times$
	$(1 - Dc)$
	}
	[*Note:* The difference between this enumerator and `hard_light` is that this tests the destination color (Dc) whereas `hard_light` tests the source color (Sc). — *end note*]

`darken`	$f(Sc, Dc) = min(Sc, Dc)$
`lighten`	$f(Sc, Dc) = max(Sc, Dc)$

`color_dodge`	$if(Dc < 1)$ {
	$f(Sc, Dc) = min(1, \frac{Dc}{(1 - Sc)})$
	}
	else {
	$f(Sc, Dc) = 1$}

`color_burn`	$if\ (Dc > 0)$ {
	$f(Sc, Dc) = 1 - min(1, \frac{1 - Dc}{Sc})$
	}
	else {
	$f(Sc, Dc) = 0$
	}

`hard_light`	$if\ (Sc \leq 0.5)$ {
	$f(Sc, Dc) = 2 \times Sc \times Dc$
	}
	else {
	$f(Sc, Dc) =$
	$1 - 2 \times (1 - Sc) \times$
	$(1 - Dc)$
	}
	[*Note:* The difference between this enumerator and `overlay` is that this tests the source color (Sc) whereas `overlay` tests the destination color (Dc). — *end note*]

Table 8 — `compositing_operator` blend enumerator meanings (continued)

Enumerator	Color
`soft_light`	$if\ (Sc \leq 0.5)\ \{$ $\quad f(Sc, Dc) =$ $\quad\quad Dc - (1 - 2 \times Sc) \times Dc \times$ $\quad\quad (1 - Dc)$ $\}$ $else\ \{$ $\quad f(Sc, Dc) =$ $\quad\quad Dc + (2 \times Sc - 1) \times$ $\quad\quad (g(Dc) - Sc)$ $\}$ $g(Dc)$ is defined as follows: $if\ (Dc \leq 0.25)\ \{$ $\quad g(Dc) =$ $\quad\quad ((16 \times Dc - 12) \times Dc +$ $\quad\quad 4) \times Dc$ $\}$ $else\ \{$ $\quad g(Dc) = \sqrt{Dc}$ $\}$
`difference`	$f(Sc, Dc) = abs(Dc - Sc)$
`exclusion`	$f(Sc, Dc) = Sc + Dc - 2 \times Sc \times Dc$

20 For the hsl enumerators, the color channels shall be treated as nonseparable, meaning that the color formula shall be evaluated once, with the colors being passed in as tuples in the form (red, green, blue).

21 The following additional functions are used to define the hsl enumerator formulas:

22 $min(x,\ y,\ z)\ =\ min(x,\ min(y,\ z))$

23 $max(x,\ y,\ z)\ =\ max(x,\ max(y,\ z))$

24 $sat(C) = max(Cr,\ Cg,\ Cb) - min(Cr,\ Cg,\ Cb)$

25 $lum(C) = Cr \times 0.3 + Cg \times 0.59 + Cb \times 0.11$

26 $clip\_color(C) =\ \{$
$\quad L = lum(C)$
$\quad N = min(Cr, Cg, Cb)$
$\quad X = max(Cr, Cg, Cb)$
$\quad if\ (N < 0.0)\ \{$
$\quad\quad Cr = L + \frac{((Cr - L) \times L)}{(L - N)}$
$\quad\quad Cg = L + \frac{((Cg - L) \times L)}{(L - N)}$
$\quad\quad Cb = L + \frac{((Cb - L) \times L)}{(L - N)}$
$\quad \}$
$\quad if\ (X > 1.0)\ \{$

$$Cr = L + \frac{((Cr - L) \times (1 - L))}{(X - L)}$$
$$Cg = L + \frac{((Cg - L) \times (1 - L))}{(X - L)}$$
$$Cb = L + \frac{((Cb - L) \times (1 - L))}{(X - L)}$$

}
return C
}

27 $set\_lum(C, L) =$ {
$D = L - lum(C)$
$Cr = Cr + D$
$Cg = Cg + D$
$Cb = Cb + D$
return $clip\_color(C)$
}

28 $set\_sat(C, S) =$ {
$R = C$
auto& $max = (Rr > Rg)\ ?\ ((Rr > Rb)\ ?\ Rr : Rb) : ((Rg > Rb)\ ?\ Rg : Rb)$
auto& $mid = (Rr > Rg)\ ?\ ((Rr > Rb)\ ?\ ((Rg > Rb)\ ?\ Rg : Rb) : Rr) : ((Rg > Rb)\ ?\ ((Rr > Rb)\ ?\ Rr : Rb) : Rg)$
auto& $min = (Rr > Rg)\ ?\ ((Rg > Rb)\ ?\ Rb : Rg) : ((Rr > Rb)\ ?\ Rb : Rr)$
if $(max > min)$ {
$$mid = \frac{((mid - min) \times S)}{max - min}$$
$max = S$
}
else {
$mid = 0.0$
$max = 0.0$
}
$min = 0.0$
return R
} [*Note:* In the formula, max, mid, and min are reference variables which are bound to the highest value, second highest value, and lowest value color channels of the (red, blue, green) tuple R such that the subsequent operations modify the values of R directly. — *end note*]

Table 9 — `compositing_operator` hsl enumerator meanings

Enumerator	Color & Alpha
`hsl_hue`	$f(Sc, Dc) = set\_lum(set\_sat(Sc,\ sat(Dc)),\ lum(Dc))$
`hsl_saturation`	$(Sc, Dc) = set\_lum(set\_sat(Dc, sat(Sc)),\ lum(Dc))$
`hsl_color`	$f(Sc, Dc) = set\_lum(Sc,\ lum(Dc))$
`hsl_luminosity`	$f(Sc, Dc) = set\_lum(Dc,\ lum(Sc))$

10 Enum class `format` [io2d.format]

10.1 `format` Summary [io2d.format.summary]

1 The `format` enum class describes the data format of the pixels of an `image_surface`. See Table 10 for the meaning of each `format` enumerator.

2 Unless otherwise specified, each channel of a pixel shall be treated as an unsigned integral value.

3 A channel value of 0x0 means that there is no contribution from that channel. As the channel value increases towards the maximum unsigned integral value representable by the number of bits of the channel, the contribution from that channel also increases. [*Example:* Given a 5-bit red channel, a value of 0x0 means that the red channel does not contribute any value towards the final color of the pixel. A value of 0x1F means that the red channel makes its maximum contribution to the final color of the pixel. — *end example*]

10.2 `format` Synopsis [io2d.format.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class format {
    invalid,
    argb32,
    rgb24,
    a8,
    rgb16_565,
    rgb30
  };
} } } } // namespaces std::experimental::graphics::v1
```

10.3 `format` Enumerators [io2d.format.enumerators]

Table 10 — `format` enumerator meanings

Enumerator	Meaning
`invalid`	A previously specified `format` is unsupported by the implementation.
`argb32`	A 32-bit pixel format. The upper 8 bits are an alpha channel, followed by an 8-bit red color channel, then an 8-bit green color channel, and finally an 8-bit blue color channel. The value in each channel is an unsigned normalized integer. The endianness of the 32-bit value is implementation-defined. This is a premultiplied format. [*Note:* It is valid for an implementation to define the endianness of the 32-bit value as being native-endian. — *end note*]
`rgb24`	A 32-bit pixel format. The upper 8 bits are unused, followed by an 8-bit red color channel, then an 8-bit green channel, and finally an 8-bit blue channel. The value in each channel is an unsigned normalized integer.
`a8`	An 8-bit pixel format composed of an alpha channel. The value in the channel is an unsigned normalized integer.

Table 10 — `format` enumerator meanings (continued)

Enumerator	Meaning
`rgb16_565`	A 16-bit pixel format composed on a red channel in the upper 5 bits, followed by a 6-bit green channel, and finally a 5-bit blue channel. The value in each channel is an unsigned normalized integer.
`rgb30`	A 32-bit pixel format. The upper 2 bits are unused, followed by a 10-bit red channel, a 10-bit green channel, and finally a 10-bit blue channel. The value in each channel is an unsigned normalized integer.

11 Enum class `path_data_type` [io2d.pathdatatype]

11.1 `path_data_type` Summary [io2d.pathdatatype.summary]

1 The `path_data_type` enum class specifies the type of path data contained by a `path_data` object. This determines which member of `path_data::data` should be used. See Table 11 for the meaning of each `path_data_type` enumerator.

11.2 `path_data_type` Synopsis [io2d.pathdatatype.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class path_data_type {
    move_to,
    line_to,
    curve_to,
    new_sub_path,
    close_path,
    rel_move_to,
    rel_line_to,
    rel_curve_to,
    arc,
    arc_negative,
    change_matrix,
    change_origin
  };
} } } } // namespaces std::experimental::io2d::v1
```

11.3 `path_data_type` Enumerators [io2d.pathdatatype.enumerators]

1 For the following table, assume that there is a `path_data p` object. The *Meaning* column describes the operation type while the *Data Member* column specifies the correct `p.data` member to access when `p.type` is equal to the enumerator.

Table 11 — `path_data_type` enumerator meanings

Enumerator	Meaning	Data Member
`move_to`	Set the *current point* and the *last move to point*. This starts a new sub-path.	`p.data.move`
`line_to`	Create a line from the `current point` to `p.data.line`. Make the new point the *current point*.	`p.data.line`

Table 11 — `path_data_type` enumerator meanings (continued)

Enumerator	Meaning	Data Member
`curve_to`	Create a cubic Bézier curve. The start point is the *current point*. The first control point is `p.data.curve.pt1`. The second control point is `p.data.curve.pt2`. The end point is `p.data.curve.pt3`. Set the *current point* to be equal to `p.data.curve.pt3`.	`p.data.curve`
`new_sub_path`	Create a new sub-path. There is no `current point` or `last move to point` after this operation.	`p.data.unused`
`close_path`	If there is a *current point*, close the *current path* by creating a line from the `current point` to the `last move to point` [*Note:* this differs from `line_to` in that the line is joined to the initial path segment (instead of each end having line caps) — *end note*], then set the *current point* to the value of the *last move to point*. If there is no *current point* then do nothing.	`p.data.unused`
`rel_move_to`	Adds the point in `p.data.move` to the *current point* and then sets the *current point* and the *last move to point* to be equal to the resulting point.	`p.data.move`
`rel_line_to`	Adds the point in `p.data.line` to the *current point* and creates a line from the *current point* to the resulting point. Make the resulting point the new *current point*.	`p.data.line`
`rel_curve_to`	Create a Bézier curve. The start point is the *current point*. The first control point is the point created by adding *current point* and `p.data.curve.pt1`. The second control point is the point created by adding *current point* and `p.data.curve.pt2`. The end point is the point created by adding *current point* and `p.data.curve.pt3`. Make the point created by adding *current point* and `p.data.curve.pt3` the new `current point`.	`p.data.curve`

Table 11 — `path_data_type` enumerator meanings (continued)

Enumerator	Meaning	Data Member
`arc`	Create a circular arc. [*Note:* The transformation matrix is used to modify a circular arc to create an elliptical arc. — *end note*] The center of the arc is `p.data.arc.center`. Its radius is `p.data.arc.radius`. It starts at the point at `p.data.arc.angle1` radians which is `p.data.arc.radius` units from `p.data.arc.center` and proceeds clockwise to the point at `p.data.arc.angle2` radians which is `p.data.arc.radius` units from `p.data.arc.center`. If `p.data.arc.angle2` is less that `p.data.arc.angle1`, continually increase `p.data.arc.angle2` by $2 \times \pi$ until it is equal to or greater than `p.data.arc.angle1`.	`p.data.arc`
`arc_negative`	Create a circular arc. [*Note:* The transformation matrix is used to modify a circular arc to create an elliptical arc. — *end note*] The center of the arc is `p.data.arc.center`. Its radius is `p.data.arc.radius`. It starts at the point at `p.data.arc.angle1` radians which is `p.data.arc.radius` units from `p.data.arc.center` and proceeds counterclockwise to the point at `p.data.arc.angle2` radians which is `p.data.arc.radius` units from `p.data.arc.center`. If `p.data.arc.angle2` is greater that `p.data.arc.angle1`, continually decrease `p.data.arc.angle2` by $2 \times \pi$ until it is equal to or less than `p.data.arc.angle1`.	`p.data.arc`
`change_matrix`	Establish a new transformation matrix.	`p.data.matrix`

Table 11 — `path_data_type` enumerator meanings (continued)

Enumerator	Meaning	Data Member
`change_origin`	Establish a new origin point. [*Note:* When a point is transformed by the transformation matrix, the origin is subtracted from it before it is transformed and then added back to the result of the transformation. — *end note*]	`p.data.origin`

12 Enum class `extend` [io2d.extend]

12.1 `extend` Summary [io2d.extend.summary]

1 The `extend` enum class describes how a pixel's color should be determined if it is outside the boundary of the source `pattern` or `surface` during a call to `surface::fill`, `surface::mask`, `surface::paint`, or `stroke`. See Table 12 for the meaning of each `extend` enumerator.

12.2 `extend` Synopsis [io2d.extend.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class extend {
    none,
    repeat,
    reflect,
    pad,
    default_extend = none
  };
} } } } // namespaces std::experimental::io2d::v1
```

12.3 `extend` Enumerators [io2d.extend.enumerators]

Table 12 — `extend` enumerator meanings

Enumerator	Meaning
`none`	Any pixel outside of the bounds of the `pattern` or `surface` which must be set will be set to transparent black.
`repeat`	A pixel outside of the bounds of the `pattern` or `surface` which must be set will be set to the color it would have been set to if the `pattern` or `surface` was infinitely large and repeated itself in a left-to-right-left-to-right and top-to-bottom-top-to-bottom fashion.
`reflect`	A pixel outside of the bounds of the `pattern` or `surface` which must be set will be set to the color it would have been set to if the `pattern` or `surface` was infinitely large and repeated itself in a left-to-right-to-left-to-right and top-to-bottom-to-top-to-bottom fashion.
`pad`	A pixel outside of the bounds of the `pattern` or `surface` which must be set will be set to the color of the nearest pixel that was in bounds.

13 Enum class `filter` [io2d.filter]

13.1 `filter` Summary [io2d.filter.summary]

1 The `filter` enum class specifies the type of signal filter to use when sampling from a `pattern` or `surface`. Some values specify attributes, leaving the choice of specific filter to the implementation, while others denote that a specific type of filter should be used. See Table 13 for the meaning of each `filter` enumerator.

13.2 `filter` Synopsis [io2d.filter.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class filter {
    fast,
    good,
    best,
    nearest,
    bilinear,
    default_filter = good
  };
} } } } // namespaces std::experimental::io2d::v1
```

13.3 `filter` Enumerators [io2d.filter.enumerators]

Table 13 — `filter` enumerator meanings

Enumerator	Meaning
`fast`	implementation-defined [*Note:* By choosing this value, the user is hinting that performance is more important that quality. Typically this would mean nearest-neighbor filtering, but if implementations can perform, e.g., bilinear filtering at nearly identical speeds, they are free to set that as the fast filter. — *end note*]
`good`	implementation-defined [*Note:* By choosing this value, the user is hinting that quality is important but that performance is also still a consideration. Typically this would mean bilinear filtering, but if implementations can perform, e.g., mipmapped bilinear filtering at nearly identical speeds, they are free to set that as the fast filter. — *end note*]
`best`	implementation-defined [*Note:* By choosing this value, the user is hinting that quality is more important than performance. Typically this would mean mipmapped bilinear filtering or mipmapped bicubic filtering. — *end note*]
`nearest`	Nearest-neighbor filtering shall be used. [*Note:* This is sometimes also called point sampling. The color of the nearest source pixel is used with no interpolation performed. Whether mipmapping is used and whether mipmaps are generated for sources that do not have them is implementation-defined. — *end note*]

Table 13 — `filter` enumerator meanings (continued)

Enumerator	Meaning
`bilinear`	Bilinear filtering shall be used. [*Note:* The distance-weighted average of the nearest 2x2 grid of source pixels is used to create an interpolated color for the destination pixel. If some source pixel values do not exist (e.g. because an edge or corner of the source has been reached) then the current `extend` should be considered in order to determine the values for the missing source pixels. If a hardware sampler is available and offers bilinear filtering, implementations may use it even if it does not conform to the description of bilinear filtering provided by this standard. Whether mipmapping is used and whether mipmaps are generated for sources that do not have them is implementation-defined. — *end note*]

14 Enum class `pattern_type` [io2d.pattern.type]

14.1 `pattern_type` Summary [io2d.pattern.type.summary]

1 The `pattern_type` enum class denotes a `pattern` object's type. [*Note:* Since a `pattern` can only be formed using one of the pattern builder types, this enum class is described in terms of which pattern builder was used to form the `pattern` which has a particular `pattern_type`. — *end note*] See Table 14 for the meaning of each `pattern_type` enumerator.

14.2 `pattern_type` Synopsis [io2d.pattern.type.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class pattern_type {
    solid_color = 1,
    linear,
    radial,
    mesh
  };
} } } } // namespaces std::experimental::io2d::v1
```

14.3 `pattern_type` Enumerators [io2d.pattern.type.enumerators]

Table 14 — `pattern_type` enumerator meanings

Enumerator	Meaning
`solid_color`	The pattern was formed using a `solid_color_pattern_builder` object.
`linear`	The pattern was formed using a `linear_pattern_builder` object.
`radial`	The pattern was formed using a `radial_pattern_builder` object.
`mesh`	The pattern was formed using a `mesh_pattern_builder` object.

15 Enum class `font_slant` [io2d.fontslant]

15.1 `font_slant` Summary [io2d.fontslant.summary]

1 The `font_slant` enum class specifies the slant requested for rendering text. [*Note:* These values have different meanings for different scripts. For some scripts they may have no meaning at all. Further, not all typefaces will support every value. As such, the values are requests which should be honored if possible and meaningful, not requirements. — *end note*] See Table 15 for the meaning of each `font_slant` enumerator.

15.2 `font_slant` Synopsis [io2d.fontslant.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class font_slant {
    normal,
    italic,
    oblique
  };
} } } } // namespaces std::experimental::io2d::v1
```

15.3 `font_slant` Enumerators [io2d.fontslant.enumerators]

Table 15 — `font_slant` enumerator meanings

Enumerator	Meaning
`normal`	The text should be rendered in whatever is a normal type for the script.
`italic`	The text should be rendered in whatever is an italic type for the script. [*Note:* If a font does not have an italic type but does have an oblique type, the oblique type shall be used if `italic` is requested. — *end note*]
`oblique`	The text should be rendered in whatever is an oblique type for the script. [*Note:* If a font does not have an oblique type but does have an italic type, the italic type shall be used if `oblique` is requested. — *end note*]

16 Enum class `font_weight` [io2d.font.weight]

16.1 `font_weight` Summary [io2d.font.weight.summary]

1 The `font_weight` enum class specifies the weight requested for rendering text. See Table 16 for the meaning of each enumerator.

16.2 `font_weight` Synopsis [io2d.font.weight.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class font_weight {
    normal,
    bold
  };
} } } } // namespaces std::experimental::io2d::v1
```

16.3 `font_weight` Enumerators [io2d.font.weight.enumerators]

Table 16 — `font_weight` enumerator meanings

Enumerator	Meaning
`normal`	The text should be rendered in whatever is a normal weight for the script.
`bold`	The text should be rendered in whatever is a bold weight for the script.

17 Enum class `subpixel_order` [io2d.subpixel.order]

17.1 `subpixel_order` Summary [io2d.subpixel.order.summary]

1 The `subpixel_order` enum class is used to request a specific order of color channels for each pixel of an output device. When a `surface` object's `font_options` object has its `antialias` value set to `antialias::subpixel` and its `subpixel_order` value set to one of these values, an implementation should use the specified `subpixel_order` to render text. See Table 17 for the meaning of each `subpixel_order` enumerator.

17.2 `subpixel_order` Synopsis [io2d.subpixel.order.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class subpixel_order {
    default_subpixel_order,
    horizontal_rgb,
    horizontal_bgr,
    vertical_rgb,
    vertical_bgr
  };
} } } } // namespaces std::experimental::io2d::v1
```

17.3 `subpixel_order` Enumerators [io2d.subpixel.order.enumerators]

Table 17 — `subpixel_order` enumerator meanings

Enumerator	Meaning
`default_subpixel_order`	The implementation should use the target `surface` object's default subpixel order.
`horizontal_rgb`	The color channels should be arranged horizontally starting with red on the left, followed by green, then blue.
`horizontal_bgr`	The color channels should be arranged horizontally starting with blue on the left, followed by green, then red.
`vertical_rgb`	The color channels should be arranged vertically starting with red on the top, followed by green, then blue.
`vertical_bgr`	The color channels should be arranged vertically starting with blue on the top, followed by green, then red.

18 Struct `rgba_color` [io2d.rgbacolor]

18.1 `rgba_color` Description [io2d.rgbacolor.intro]

1 The `rgba_color` struct represents a premultiplied color with four channels.

2 There are three color channels, each of which is an unsigned normalized double: red, green, and blue.

3 There is also an alpha channel, which is an unsigned normalized double.

4 The use of an unsigned normalized double allows for storing unsigned normalized integer values with easy, albeit potentially lossy, conversion between different bits per channel (e.g. storing a 32-bit X8R8G8B8 value and retrieving its nearest 16-bit equivalent R5G6B5 value). More importantly, use of a double ensures sufficient precision for lossless storage and retrieval, without conversion, of 32-bit color formats and even 64-bit (16-bits per channel) color formats.

18.2 `rgba_color` synopsis [io2d.rgbacolor.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  struct rgba_color {
    double r;
    double g;
    double b;
    double a;

    // 18.3, static constants:
    const static rgba_color alice_blue;
    const static rgba_color antique_white;
    const static rgba_color aqua;
    const static rgba_color aquamarine;
    const static rgba_color azure;
    const static rgba_color beige;
    const static rgba_color bisque;
    const static rgba_color black;
    const static rgba_color blanched_almond;
    const static rgba_color blue;
    const static rgba_color blue_violet;
    const static rgba_color brown;
    const static rgba_color burly_wood;
    const static rgba_color cadet_blue;
    const static rgba_color chartreuse;
    const static rgba_color chocolate;
    const static rgba_color coral;
    const static rgba_color cornflower_blue;
    const static rgba_color cornsilk;
    const static rgba_color crimson;
    const static rgba_color cyan;
    const static rgba_color dark_blue;
    const static rgba_color dark_cyan;
    const static rgba_color dark_goldenrod;
    const static rgba_color dark_gray;
    const static rgba_color dark_green;
    const static rgba_color dark_grey;
    const static rgba_color dark_khaki;
```

```
const static rgba_color dark_magenta;
const static rgba_color dark_olive_green;
const static rgba_color dark_orange;
const static rgba_color dark_orchid;
const static rgba_color dark_red;
const static rgba_color dark_salmon;
const static rgba_color dark_sea_green;
const static rgba_color dark_slate_blue;
const static rgba_color dark_slate_gray;
const static rgba_color dark_slate_grey;
const static rgba_color dark_turquoise;
const static rgba_color dark_violet;
const static rgba_color deep_pink;
const static rgba_color deep_sky_blue;
const static rgba_color dim_gray;
const static rgba_color dim_grey;
const static rgba_color dodger_blue;
const static rgba_color firebrick;
const static rgba_color floral_white;
const static rgba_color forest_green;
const static rgba_color fuchsia;
const static rgba_color gainsboro;
const static rgba_color ghost_white;
const static rgba_color gold;
const static rgba_color goldenrod;
const static rgba_color gray;
const static rgba_color green;
const static rgba_color green_yellow;
const static rgba_color grey;
const static rgba_color honeydew;
const static rgba_color hot_pink;
const static rgba_color indian_red;
const static rgba_color indigo;
const static rgba_color ivory;
const static rgba_color khaki;
const static rgba_color lavender;
const static rgba_color lavender_blush;
const static rgba_color lawn_green;
const static rgba_color lemon_chiffon;
const static rgba_color light_blue;
const static rgba_color light_coral;
const static rgba_color light_cyan;
const static rgba_color light_goldenrod_yellow;
const static rgba_color light_gray;
const static rgba_color light_green;
const static rgba_color light_grey;
const static rgba_color light_pink;
const static rgba_color light_salmon;
const static rgba_color light_sea_green;
const static rgba_color light_sky_blue;
const static rgba_color light_slate_gray;
const static rgba_color light_slate_grey;
const static rgba_color light_steel_blue;
const static rgba_color light_yellow;
const static rgba_color lime;
```

```
const static rgba_color lime_green;
const static rgba_color linen;
const static rgba_color magenta;
const static rgba_color maroon;
const static rgba_color medium_aquamarine;
const static rgba_color medium_blue;
const static rgba_color medium_orchid;
const static rgba_color medium_purple;
const static rgba_color medium_sea_green;
const static rgba_color medium_slate_blue;
const static rgba_color medium_spring_green;
const static rgba_color medium_turquoise;
const static rgba_color medium_violet_red;
const static rgba_color midnight_blue;
const static rgba_color mint_cream;
const static rgba_color misty_rose;
const static rgba_color moccasin;
const static rgba_color navajo_white;
const static rgba_color navy;
const static rgba_color old_lace;
const static rgba_color olive;
const static rgba_color olive_drab;
const static rgba_color orange;
const static rgba_color orange_red;
const static rgba_color orchid;
const static rgba_color pale_goldenrod;
const static rgba_color pale_green;
const static rgba_color pale_turquoise;
const static rgba_color pale_violet_red;
const static rgba_color papaya_whip;
const static rgba_color peach_puff;
const static rgba_color peru;
const static rgba_color pink;
const static rgba_color plum;
const static rgba_color powder_blue;
const static rgba_color purple;
const static rgba_color red;
const static rgba_color rosy_brown;
const static rgba_color royal_blue;
const static rgba_color saddle_brown;
const static rgba_color salmon;
const static rgba_color sandy_brown;
const static rgba_color sea_green;
const static rgba_color sea_shell;
const static rgba_color sienna;
const static rgba_color silver;
const static rgba_color sky_blue;
const static rgba_color slate_blue;
const static rgba_color slate_gray;
const static rgba_color slate_grey;
const static rgba_color snow;
const static rgba_color spring_green;
const static rgba_color steel_blue;
const static rgba_color tan;
const static rgba_color teal;
```

```
    const static rgba_color thistle;
    const static rgba_color tomato;
    const static rgba_color transparent_black;
    const static rgba_color turquoise;
    const static rgba_color violet;
    const static rgba_color wheat;
    const static rgba_color white;
    const static rgba_color white_smoke;
    const static rgba_color yellow;
    const static rgba_color yellow_green;
  };

  // 18.4, rgba_color non-member functions:
  rgba_color operator*(const rgba_color& lhs, double rhs);
  rgba_color& operator*=(rgba_color& lhs, double rhs);
  bool operator==(const rgba_color& lhs, const rgba_color& rhs);
  bool operator!=(const rgba_color& lhs, const rgba_color& rhs);
} } } } // namespaces std::experimental::io2d::v1
```

18.3 `rgba_color` static constants [io2d.rgbacolor.constants]

Table 18 — `rgba_color` const static member values

Constant	Value
alice_blue	rgba_color{ 240ubyte, 248ubyte, 255ubyte, 255ubyte }
antique_white	rgba_color{ 250ubyte, 235ubyte, 215ubyte, 255ubyte }
aqua	rgba_color{ 0ubyte, 255ubyte, 255ubyte, 255ubyte }
aquamarine	rgba_color{ 127ubyte, 255ubyte, 212ubyte, 255ubyte }
azure	rgba_color{ 240ubyte, 255ubyte, 255ubyte, 255ubyte }
beige	rgba_color{ 245ubyte, 245ubyte, 220ubyte, 255ubyte }
bisque	rgba_color{ 255ubyte, 228ubyte, 196ubyte, 255ubyte }
black	rgba_color{ 0ubyte, 0ubyte, 0ubyte, 255ubyte }
blanched_almond	rgba_color{ 255ubyte, 235ubyte, 205ubyte, 255ubyte }
blue	rgba_color{ 0ubyte, 0ubyte, 255ubyte, 255ubyte }
blue_violet	rgba_color{ 138ubyte, 43ubyte, 226ubyte, 255ubyte }
brown	rgba_color{ 165ubyte, 42ubyte, 42ubyte, 255ubyte }
burly_wood	rgba_color{ 222ubyte, 184ubyte, 135ubyte, 255ubyte }
cadet_blue	rgba_color{ 95ubyte, 158ubyte, 160ubyte, 255ubyte }
chartreuse	rgba_color{ 127ubyte, 255ubyte, 0ubyte, 255ubyte }

Table 18 — `rgba_color` const static member values (continued)

Constant	Value
chocolate	rgba_color{ 210ubyte, 105ubyte, 30ubyte, 255ubyte }
coral	rgba_color{ 255ubyte, 127ubyte, 80ubyte, 255ubyte }
cornflower_blue	rgba_color{ 100ubyte, 149ubyte, 237ubyte, 255ubyte }
cornsilk	rgba_color{ 255ubyte, 248ubyte, 220ubyte, 255ubyte }
crimson	rgba_color{ 220ubyte, 20ubyte, 60ubyte, 255ubyte }
cyan	rgba_color{ 0ubyte, 255ubyte, 255ubyte, 255ubyte }
dark_blue	rgba_color{ 0ubyte, 0ubyte, 139ubyte, 255ubyte }
dark_cyan	rgba_color{ 0ubyte, 139ubyte, 139ubyte, 255ubyte }
dark_goldenrod	rgba_color{ 184ubyte, 134ubyte, 11ubyte, 255ubyte }
dark_gray	rgba_color{ 169ubyte, 169ubyte, 169ubyte, 255ubyte }
dark_green	rgba_color{ 0ubyte, 100ubyte, 0ubyte, 255ubyte }
dark_grey	rgba_color{ 169ubyte, 169ubyte, 169ubyte, 255ubyte }
dark_khaki	rgba_color{ 189ubyte, 183ubyte, 107ubyte, 255ubyte }
dark_magenta	rgba_color{ 139ubyte, 0ubyte, 139ubyte, 255ubyte }
dark_olive_green	rgba_color{ 85ubyte, 107ubyte, 47ubyte, 255ubyte }
dark_orange	rgba_color{ 255ubyte, 140ubyte, 0ubyte, 255ubyte }
dark_orchid	rgba_color{ 153ubyte, 50ubyte, 204ubyte, 255ubyte }
dark_red	rgba_color{ 139ubyte, 0ubyte, 0ubyte, 255ubyte }
dark_salmon	rgba_color{ 233ubyte, 150ubyte, 122ubyte, 255ubyte }
dark_sea_green	rgba_color{ 143ubyte, 188ubyte, 143ubyte, 255ubyte }
dark_slate_blue	rgba_color{ 72ubyte, 61ubyte, 139ubyte, 255ubyte }
dark_slate_gray	rgba_color{ 47ubyte, 79ubyte, 79ubyte, 255ubyte }
dark_slate_grey	rgba_color{ 47ubyte, 79ubyte, 79ubyte, 255ubyte }
dark_turquoise	rgba_color{ 0ubyte, 206ubyte, 209ubyte, 255ubyte }
dark_violet	rgba_color{ 148ubyte, 0ubyte, 211ubyte, 255ubyte }
deep_pink	rgba_color{ 255ubyte, 20ubyte, 147ubyte, 255ubyte }

Table 18 — `rgba_color` const static member values (continued)

Constant	Value
deep_sky_blue	rgba_color{ 0ubyte, 191ubyte, 255ubyte, 255ubyte }
dim_gray	rgba_color{ 105ubyte, 105ubyte, 105ubyte, 255ubyte }
dim_grey	rgba_color{ 105ubyte, 105ubyte, 105ubyte, 255ubyte }
dodger_blue	rgba_color{ 30ubyte, 144ubyte, 255ubyte, 255ubyte }
firebrick	rgba_color{ 178ubyte, 34ubyte, 34ubyte, 255ubyte }
floral_white	rgba_color{ 255ubyte, 250ubyte, 240ubyte, 255ubyte }
forest_green	rgba_color{ 34ubyte, 139ubyte, 34ubyte, 255ubyte }
fuchsia	rgba_color{ 255ubyte, 0ubyte, 255ubyte, 255ubyte }
gainsboro	rgba_color{ 220ubyte, 220ubyte, 220ubyte, 255ubyte }
ghost_white	rgba_color{ 248ubyte, 248ubyte, 255ubyte, 255ubyte }
gold	rgba_color{ 255ubyte, 215ubyte, 0ubyte, 255ubyte }
goldenrod	rgba_color{ 218ubyte, 165ubyte, 32ubyte, 255ubyte }
gray	rgba_color{ 128ubyte, 128ubyte, 128ubyte, 255ubyte }
green	rgba_color{ 0ubyte, 128ubyte, 0ubyte, 255ubyte }
green_yellow	rgba_color{ 173ubyte, 255ubyte, 47ubyte, 255ubyte }
grey	rgba_color{ 128ubyte, 128ubyte, 128ubyte, 255ubyte }
honeydew	rgba_color{ 240ubyte, 255ubyte, 240ubyte, 255ubyte }
hot_pink	rgba_color{ 255ubyte, 105ubyte, 180ubyte, 255ubyte }
indian_red	rgba_color{ 205ubyte, 92ubyte, 92ubyte, 255ubyte }
indigo	rgba_color{ 75ubyte, 0ubyte, 130ubyte, 255ubyte }
ivory	rgba_color{ 255ubyte, 255ubyte, 240ubyte, 255ubyte }
khaki	rgba_color{ 240ubyte, 230ubyte, 140ubyte, 255ubyte }
lavender	rgba_color{ 230ubyte, 230ubyte, 250ubyte, 255ubyte }
lavender_blush	rgba_color{ 255ubyte, 240ubyte, 245ubyte, 255ubyte }
lawn_green	rgba_color{ 124ubyte, 252ubyte, 0ubyte, 255ubyte }

Table 18 — `rgba_color` const static member values (continued)

Constant	Value
`lemon_chiffon`	`rgba_color{ 255ubyte, 250ubyte, 205ubyte, 255ubyte }`
`light_blue`	`rgba_color{ 173ubyte, 216ubyte, 230ubyte, 255ubyte }`
`light_coral`	`rgba_color{ 240ubyte, 128ubyte, 128ubyte, 255ubyte }`
`light_cyan`	`rgba_color{ 224ubyte, 255ubyte, 255ubyte, 255ubyte }`
`light_goldenrod_yellow`	`rgba_color{ 250ubyte, 250ubyte, 210ubyte, 255ubyte }`
`light_gray`	`rgba_color{ 211ubyte, 211ubyte, 211ubyte, 255ubyte }`
`light_green`	`rgba_color{ 144ubyte, 238ubyte, 144ubyte, 255ubyte }`
`light_grey`	`rgba_color{ 211ubyte, 211ubyte, 211ubyte, 255ubyte }`
`light_pink`	`rgba_color{ 255ubyte, 182ubyte, 193ubyte, 255ubyte }`
`light_salmon`	`rgba_color{ 255ubyte, 160ubyte, 122ubyte, 255ubyte }`
`light_sea_green`	`rgba_color{ 32ubyte, 178ubyte, 170ubyte, 255ubyte }`
`light_sky_blue`	`rgba_color{ 135ubyte, 206ubyte, 250ubyte, 255ubyte }`
`light_slate_gray`	`rgba_color{ 119ubyte, 136ubyte, 153ubyte, 255ubyte }`
`light_slate_grey`	`rgba_color{ 119ubyte, 136ubyte, 153ubyte, 255ubyte }`
`light_steel_blue`	`rgba_color{ 176ubyte, 196ubyte, 222ubyte, 255ubyte }`
`light_yellow`	`rgba_color{ 255ubyte, 255ubyte, 224ubyte, 255ubyte }`
`lime`	`rgba_color{ 0ubyte, 255ubyte, 0ubyte, 255ubyte }`
`lime_green`	`rgba_color{ 50ubyte, 205ubyte, 50ubyte, 255ubyte }`
`linen`	`rgba_color{ 250ubyte, 240ubyte, 230ubyte, 255ubyte }`
`magenta`	`rgba_color{ 255ubyte, 0ubyte, 255ubyte, 255ubyte }`
`maroon`	`rgba_color{ 128ubyte, 0ubyte, 0ubyte, 255ubyte }`
`medium_aquamarine`	`rgba_color{ 102ubyte, 205ubyte, 170ubyte, 255ubyte }`
`medium_blue`	`rgba_color{ 0ubyte, 0ubyte, 205ubyte, 255ubyte }`
`medium_orchid`	`rgba_color{ 186ubyte, 85ubyte, 211ubyte, 255ubyte }`
`medium_purple`	`rgba_color{ 147ubyte, 112ubyte, 219ubyte, 255ubyte }`

Table 18 — `rgba_color` const static member values (continued)

Constant	Value
medium_sea_green	rgba_color{ 60ubyte, 179ubyte, 113ubyte, 255ubyte }
medium_slate_blue	rgba_color{ 123ubyte, 104ubyte, 238ubyte, 255ubyte }
medium_spring_green	rgba_color{ 0ubyte, 250ubyte, 154ubyte, 255ubyte }
medium_turquoise	rgba_color{ 72ubyte, 209ubyte, 204ubyte, 255ubyte }
medium_violet_red	rgba_color{ 199ubyte, 21ubyte, 133ubyte, 255ubyte }
midnight_blue	rgba_color{ 25ubyte, 25ubyte, 112ubyte, 255ubyte }
mint_cream	rgba_color{ 245ubyte, 255ubyte, 250ubyte, 255ubyte }
misty_rose	rgba_color{ 255ubyte, 228ubyte, 225ubyte, 255ubyte }
moccasin	rgba_color{ 255ubyte, 228ubyte, 181ubyte, 255ubyte }
navajo_white	rgba_color{ 255ubyte, 222ubyte, 173ubyte, 255ubyte }
navy	rgba_color{ 0ubyte, 0ubyte, 128ubyte, 255ubyte }
old_lace	rgba_color{ 253ubyte, 245ubyte, 230ubyte, 255ubyte }
olive	rgba_color{ 128ubyte, 128ubyte, 0ubyte, 255ubyte }
olive_drab	rgba_color{ 107ubyte, 142ubyte, 35ubyte, 255ubyte }
orange	rgba_color{ 255ubyte, 165ubyte, 0ubyte, 255ubyte }
orange_red	rgba_color{ 255ubyte, 69ubyte, 0ubyte, 255ubyte }
orchid	rgba_color{ 218ubyte, 112ubyte, 214ubyte, 255ubyte }
pale_goldenrod	rgba_color{ 238ubyte, 232ubyte, 170ubyte, 255ubyte }
pale_green	rgba_color{ 152ubyte, 251ubyte, 152ubyte, 255ubyte }
pale_turquoise	rgba_color{ 175ubyte, 238ubyte, 238ubyte, 255ubyte }
pale_violet_red	rgba_color{ 219ubyte, 112ubyte, 147ubyte, 255ubyte }
papaya_whip	rgba_color{ 255ubyte, 239ubyte, 213ubyte, 255ubyte }
peach_puff	rgba_color{ 255ubyte, 218ubyte, 185ubyte, 255ubyte }
peru	rgba_color{ 205ubyte, 133ubyte, 63ubyte, 255ubyte }
pink	rgba_color{ 255ubyte, 192ubyte, 203ubyte, 255ubyte }

Table 18 — `rgba_color` const static member values (continued)

Constant	Value
plum	rgba_color{ 221ubyte, 160ubyte, 221ubyte, 255ubyte }
powder_blue	rgba_color{ 176ubyte, 224ubyte, 230ubyte, 255ubyte }
purple	rgba_color{ 128ubyte, 0ubyte, 128ubyte, 255ubyte }
red	rgba_color{ 255ubyte, 0ubyte, 0ubyte, 255ubyte }
rosy_brown	rgba_color{ 188ubyte, 143ubyte, 143ubyte, 255ubyte }
royal_blue	rgba_color{ 65ubyte, 105ubyte, 225ubyte, 255ubyte }
saddle_brown	rgba_color{ 139ubyte, 69ubyte, 19ubyte, 255ubyte }
salmon	rgba_color{ 250ubyte, 128ubyte, 114ubyte, 255ubyte }
sandy_brown	rgba_color{ 244ubyte, 164ubyte, 96ubyte, 255ubyte }
sea_green	rgba_color{ 46ubyte, 139ubyte, 87ubyte, 255ubyte }
sea_shell	rgba_color{ 255ubyte, 245ubyte, 238ubyte, 255ubyte }
sienna	rgba_color{ 160ubyte, 82ubyte, 45ubyte, 255ubyte }
silver	rgba_color{ 192ubyte, 192ubyte, 192ubyte, 255ubyte }
sky_blue	rgba_color{ 135ubyte, 206ubyte, 235ubyte, 255ubyte }
slate_blue	rgba_color{ 106ubyte, 90ubyte, 205ubyte, 255ubyte }
slate_gray	rgba_color{ 112ubyte, 128ubyte, 144ubyte, 255ubyte }
slate_grey	rgba_color{ 112ubyte, 128ubyte, 144ubyte, 255ubyte }
snow	rgba_color{ 255ubyte, 250ubyte, 250ubyte, 255ubyte }
spring_green	rgba_color{ 0ubyte, 255ubyte, 127ubyte, 255ubyte }
steel_blue	rgba_color{ 70ubyte, 130ubyte, 180ubyte, 255ubyte }
tan	rgba_color{ 210ubyte, 180ubyte, 140ubyte, 255ubyte }
teal	rgba_color{ 0ubyte, 128ubyte, 128ubyte, 255ubyte }
thistle	rgba_color{ 216ubyte, 191ubyte, 216ubyte, 255ubyte }
tomato	rgba_color{ 255ubyte, 99ubyte, 71ubyte, 255ubyte }

Table 18 — `rgba_color` const static member values (continued)

Constant	Value
transparent_black	rgba_color{ 0ubyte, 0ubyte, 0ubyte, 255ubyte }
turquoise	rgba_color{ 64ubyte, 244ubyte, 208ubyte, 255ubyte }
violet	rgba_color{ 238ubyte, 130ubyte, 238ubyte, 255ubyte }
wheat	rgba_color{ 245ubyte, 222ubyte, 179ubyte, 255ubyte }
white	rgba_color{ 255ubyte, 255ubyte, 255ubyte, 255ubyte }
white_smoke	rgba_color{ 245ubyte, 245ubyte, 245ubyte, 255ubyte }
yellow	rgba_color{ 255ubyte, 255ubyte, 0ubyte, 255ubyte }
yellow_green	rgba_color{ 154ubyte, 205ubyte, 50ubyte, 255ubyte }

18.4 `rgba_color` non-member functions [io2d.rgbacolor.nonmembers]

```
rgba_color operator*(const rgba_color& lhs, double rhs);
```

1 *Returns:*
```
rgba_color{
::std::max(0.0, ::std::min(1.0, lhs.r * rhs)),
::std::max(0.0, ::std::min(1.0, lhs.g * rhs)),
::std::max(0.0, ::std::min(1.0, lhs.b * rhs)),
::std::max(0.0, ::std::min(1.0, rhs))
}
```

```
rgba_color& operator*=(rgba_color& lhs, double rhs);
```

2 *Effects:* `lhs = lhs * rhs`

3 *Returns:* `lhs`

```
bool operator==(const rgba_color& lhs, const rgba_color& rhs);
```

4 *Returns:* `lhs.r == rhs.r && lhs.g == rhs.g && lhs.b == rhs.b && lhs.a == rhs.a`

```
bool operator!=(const rgba_color& lhs, const rgba_color& rhs);
```

5 *Returns:* `!(lhs == rhs)`

19 `literals` namespace [io2d.literals]

19.1 `literals` Synopsis [io2d.literals.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  inline namespace literals {
    double operator""ubyte(unsigned long long value);
    double operator""unorm(long double value);
  } // namespace literals
} } } } // namespaces std::experimental::io2d::v1
```

19.2 `literals` operators [io2d.literals.operators]

```
double operator""ubyte(unsigned long long value);
```

1 *Returns:* `::std::max(0.0, ::std::min(1.0, static_cast<double>(value) / 255.0))`

```
double operator""unorm(long double value);
```

2 *Returns:* `::std::nearbyint(::std::max(0.0, ::std::min(1.0, static_cast<double>(value))) * 255.0) / 255.0`

20 Struct `point` [io2d.point]

20.1 `point` Description [io2d.point.intro]

1 The `point` struct represents a point in coordinate space.

20.2 `point` synopsis [io2d.point.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  struct point {
    double x;
    double y;
  };

  // 20.3, point non-member functions:
  point operator+(const point& lhs);
  point operator+(const point& lhs, const point& rhs);
  point operator+(const point& lhs, double rhs);
  point& operator+=(point& lhs, const point& rhs);
  point& operator+=(point& lhs, double rhs);
  point operator-(const point& lhs);
  point operator-(const point& lhs, const point& rhs);
  point operator-(const point& lhs, double rhs);
  point& operator-=(point& lhs, const point& rhs);
  point& operator-=(point& lhs, double rhs);
  point operator*(const point& lhs, const point& rhs);
  point operator*(const point& lhs, double rhs);
  point& operator*=(point& lhs, const point& rhs);
  point& operator*=(point& lhs, double rhs);
  point operator/(const point& lhs, const point& rhs);
  point operator/(const point& lhs, double rhs);
  point& operator/=(point& lhs, const point& rhs);
  point& operator/=(point& lhs, double rhs);
  bool operator==(const point& lhs, const point& rhs);
  bool operator!=(const point& lhs, const point& rhs);
} } } } // namespaces std::experimental::io2d::v1
```

20.3 `point` non-member functions [io2d.point.nonmembers]

`point operator+(const point& lhs);`

1 *Returns:* `point(lhs)`

`point operator+(const point& lhs, const point& rhs);`

2 *Returns:* `point{ lhs.x + rhs.x, lhs.y + rhs.y }`

`point operator+(const point& lhs, double rhs);`

3 *Returns:* `point{ lhs.x + rhs, lhs.y + rhs }`

`point& operator+=(point& lhs, const point& rhs);`

4 *Effects:* `lhs = lhs + rhs`

5 *Returns:* `lhs`

```
point& operator+=(point& lhs, double rhs);
```

Effects: `lhs = lhs + rhs`

Returns: `lhs`

```
point operator-(const point& lhs);
```

Returns: `point{ -lhs.x, -lhs.y }`

```
point operator-(const point& lhs, const point& rhs);
```

Returns: `point{ lhs.x - rhs.x, lhs.y - rhs.y }`

```
point operator-(const point& lhs, double rhs);
```

Returns: `point{ lhs.x - rhs, lhs.y - rhs }`

```
point& operator-=(point& lhs, const point& rhs);
```

Effects: `lhs = lhs - rhs`

Returns: `lhs`

```
point& operator-=(point& lhs, double rhs);
```

Effects: `lhs = lhs - rhs`

Returns: `lhs`

```
point operator*(const point& lhs, const point& rhs);
```

Returns: `point{ lhs.x * rhs.x, lhs.y * rhs.y }`

```
point operator*(const point& lhs, double rhs);
```

Returns: `point{ lhs.x * rhs, lhs.y * rhs }`

```
point& operator*=(point& lhs, const point& rhs);
```

Effects: `lhs = lhs * rhs`

Returns: `lhs`

```
point& operator*=(point& lhs, double rhs);
```

Effects: `lhs = lhs * rhs`

Returns: `lhs`

```
point operator/(const point& lhs, const point& rhs);
```

Returns: `point{ lhs.x / rhs.x, lhs.y / rhs.y }`

```
point operator/(const point& lhs, double rhs);
```

Returns: `point{ lhs.x / rhs, lhs.y / rhs }`

```
point& operator/=(point& lhs, const point& rhs);
```

Effects: `lhs = lhs / rhs`

Returns: `lhs`

```
point& operator/=(point& lhs, double rhs);
```

25 *Effects:* `lhs = lhs / rhs`

26 *Returns:* `lhs`

```
bool operator==(const point& lhs, const point& rhs);
```

27 *Returns:* `lhs.x == rhs.x && lhs.y == rhs.y`

```
bool operator!=(const point& lhs, const point& rhs);
```

28 *Returns:* `!(lhs == rhs)`

21 Struct `matrix_2d` [io2d.matrix2d]

21.1 `matrix_2d` Description [io2d.matrix2d.intro]

1 The `matrix_2d` struct represents a two-dimensional, 3x3, row-major matrix for affine transformations.

2 The third column always has the column vector value of $[0.0, 0.0, 1.0]$ and so is not included in the data members of the matrix. The performance of any mathematical operation upon a `matrix_2d` shall be carried out as if the omitted data members were present with their prescribed values. [*Note:* If the third column's data members were included, they would be: `double m02`, which would immediately follow `m01` and would be assigned a value of `0.0` by all static factory functions21.3; `double m12`, which would immediately follow `m11` and would be assigned a value of `0.0` by all static factory functions21.3; and, `double m22`, which would immediately follow `m21` and would be assigned a value of `1.0` by all static factory functions21.3. The layout of the resulting matrix would be as such:
[[$m00$ $m01$ $m02$]]
[[$m10$ $m11$ $m12$]]
[[$m20$ $m21$ $m22$]] — *end note*]

21.2 `matrix_2d` synopsis [io2d.matrix2d.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  struct matrix_2d {
    double m00;
    double m01;
    double m10;
    double m11;
    double m20;
    double m21;

    // 21.3, static factory functions:
    static matrix_2d init_identity();
    static matrix_2d init_translate(const point& value);
    static matrix_2d init_scale(const point& value);
    static matrix_2d init_rotate(double radians);
    static matrix_2d init_shear_x(double factor);
    static matrix_2d init_shear_y(double factor);

    // 21.4, modifiers:
    matrix_2d& translate(const point& value);
    matrix_2d& scale(const point& value);
    matrix_2d& rotate(double radians);
    matrix_2d& shear_x(double factor);
    matrix_2d& shear_y(double factor);
    matrix_2d& invert();

    // 21.5, observers:
    double determinant() const;
    point transform_distance(const point& dist) const;
    point transform_point(const point& pt) const;
  };

  // 21.6, matrix_2d non-member functions:
```

```
    matrix_2d operator*(const matrix_2d& lhs, const matrix_2d& rhs);
    matrix_2d& operator*=(matrix_2d& lhs, const matrix_2d& rhs);
    bool operator==(const matrix_2d& lhs, const matrix_2d& rhs);
  } } } } // namespaces std::experimental::io2d::v1
```

21.3 matrix_2d static factory functions [io2d.matrix2d.staticfactories]

21.3.1 matrix_2d::init_identity [io2d.matrix2d::init_identity]

```
static matrix_2d init_identity();
```

1 *Returns:* `matrix{ 1.0, 0.0, 0.0, 1.0, 0.0, 0.0 }`

21.3.2 matrix_2d::init_translate [io2d.matrix2d::init_translate]

```
static matrix_2d init_translate(const point& value);
```

1 *Returns:* `matrix{ 1.0, 0.0, 0.0, 1.0, value.x, value.y }`

21.3.3 matrix_2d::init_scale [io2d.matrix2d::init_scale]

```
static matrix_2d init_scale(const point& value);
```

1 *Returns:* `matrix{ value.x, 0.0, 0.0, value.y, 0.0, 0.0 }`

21.3.4 matrix_2d::init_rotate [io2d.matrix2d::init_rotate]

```
static matrix_2d init_rotate(double radians);
```

1 *Returns:* `matrix{ cos(radians), sin(radians), -sin(radians), cos(radians), 0.0, 0.0 }`

21.3.5 matrix_2d::init_shear_x [io2d.matrix2d::init_shear_x]

```
static matrix_2d init_shear_x(double factor);
```

1 *Returns:* `matrix{ 1.0, 0.0, factor, 1.0, 0.0, 0.0 }`

21.3.6 matrix_2d::init_shear_y [io2d.matrix2d::init_shear_y]

```
static matrix_2d init_shear_y(double factor);
```

1 *Returns:* `matrix{ 1.0, factor, 0.0, 1.0, 0.0, 0.0 }`

21.4 matrix_2d modifiers [io2d.matrix2d.modifiers]

21.4.1 matrix_2d::translate [io2d.matrix2d::translate]

```
matrix_2d& translate(const point& value);
```

1 *Effects:* `*this = init_translate(value) * (*this)`

2 *Returns:* `*this`

21.4.2 matrix_2d::scale [io2d.matrix2d::scale]

```
matrix_2d& scale(const point& value);
```

1 *Effects:* `*this = init_scale(value) * (*this)`

2 *Returns:* `*this`

21.4.3 matrix_2d::rotate [io2d.matrix2d::rotate]

```
matrix_2d& rotate(double radians);
```

1 *Effects:* `*this = init_rotate(radians) * (*this)`

2 *Returns:* `*this`

21.4.4 matrix_2d::shear_x [io2d.matrix2d::shear_x]

```
matrix_2d& shear_x(double factor);
```

1 *Effects:* `*this = init_shear_x(factor) * (*this)`

2 *Returns:* `*this`

21.4.5 matrix_2d::shear_y [io2d.matrix2d::shear_y]

```
matrix_2d& shear_y(double factor);
```

1 *Effects:* `*this = init_shear_y(factor) * (*this)`

2 *Returns:* `*this`

21.4.6 matrix_2d::invert [io2d.matrix2d::invert]

```
matrix_2d& invert();
```

1 *Effects:* `*this` is inverted.

2 *Returns:* `*this`

3 *Throws:* `system_error` with an error code equivalent to `io2d_error::invalid_matrix` and an error category of type `io2d_error_category` if a `*this` is not an invertible matrix.

21.5 matrix_2d observers [io2d.matrix2d.observers]

21.5.1 matrix_2d::determinant [io2d.matrix2d::determinant]

```
double determinant() const;
```

1 *Returns:* The determinant of `*this`.

21.5.2 matrix_2d::transform_distance [io2d.matrix2d::transform_distance]

```
point transform_distance(const point& dist) const;
```

1 *Returns:* `point{ m00 * dist.x + m10 * dist.y, m01 * dist.x + m11 * dist.y }`

2 *Remarks:* This function ignores the translation component of `*this`. If the translation component, `m20` and `m21`, of `*this` is set to `(0.0, 0.0)`, the return value of this function and the return value of `transform_point(dist)` will be identical when given the same input.

21.5.3 matrix_2d::transform_point [io2d.matrix2d::transform_point]

```
point transform_point(const point& pt) const;
```

1 *Returns:* `point{ (m00 * dist.x + m10 * dist.y) + m20, (m01 * dist.x + m11 * dist.y) + m21 }`

21.6 matrix_2d non-member functions [io2d.matrix2d.nonmembers]

```
matrix_2d operator*(const matrix_2d& lhs, const matrix_2d& rhs);
```

1 *Returns:* matrix_2d{
(lhs.m00 * rhs.m00) + (lhs.m01 * rhs.m10),
(lhs.m00 * rhs.m01) + (lhs.m01 * rhs.m11),
(lhs.m10 * rhs.m00) + (lhs.m11 * rhs.m10),
(lhs.m10 * rhs.m01) + (lhs.m11 * rhs.m11),
(lhs.m20 * rhs.m00) + (lhs.m21 * rhs.m10) + lhs.m20,
(lhs.m20 * rhs.m01) + (lhs.m21 * rhs.m11) + lhs.m21
}

```
matrix_2d& operator*=(matrix_2d& lhs, const matrix_2d& rhs);
```

2 *Effects:* lhs = lhs * rhs

3 *Returns:* lhs

```
bool operator==(const matrix_2d& lhs, const matrix_2d& rhs);
```

4 *Returns:* lhs.m00 == rhs.m00 && lhs.m01 == rhs.m01 && lhs.m10 == rhs.m10 && lhs.m11 == rhs.m11 && lhs.m20 == rhs.m20 && lhs.m21 == rhs.m21

```
bool operator!=(const matrix_2d& lhs, const matrix_2d& rhs);
```

5 *Returns:* !(lhs == rhs)

22 Class `io2d_error_category` [io2d.io2derrcat]

22.1 `io2d_error_category` Description [io2d.io2derrcat.intro]

1 The `io2d_error_category` class derives from `::std::error_category` in order to provide a custom error category for use by this library.

22.2 `io2d_error_category` synopsis [io2d.io2derrcat.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class io2d_error_category : public ::std::error_category {
  public:
    // 22.3, observers:
    virtual const char* name() const noexcept override;
    virtual ::std::string message(int errVal) const override;
    virtual bool equivalent(int code,
      const ::std::error_condition& condition) const noexcept override;
    virtual bool equivalent(const ::std::error_code& ec,
      int condition) const noexcept override;
  };

  // 22.4, non-member functions:
  const ::std::error_category& io2d_category() noexcept;
} } } } // namespaces std::experimental::io2d::v1
```

22.3 `io2d_error_category` observers [io2d.io2derrcat.observers]

22.3.1 `io2d_error_category::name` [io2d.io2derrcat::name]

```
virtual const char* name() const noexcept override;
```

1 *Returns:* A pointer to the string `"io2d"`.

22.3.2 `io2d_error_category::message` [io2d.io2derrcat::message]

```
virtual ::std::string message(int errVal) const override;
```

1 *Returns:* When `errVal` has the same value as the integer value of an `io2d_error` enumerator, the corresponding meaning text in Table 1 shall be part of the `string` returned by this function for that value. If there is no corresponding enumerator, the return value is implementation-defined. [*Note:* When `errVal` has the same value as the integer value of an `io2d_error` enumerator, implementations should include any additional meaningful diagnostic information in the `string` returned by this function. When no equivalent value enumerator exists, implementations should return string diagnostic information provided by the underlying rendering and presentation technologies as well as any additional meaningful diagnostic information in the `string` returned by this function. — *end note*]

22.3.3 `io2d_error_category::equivalent` [io2d.io2derrcat::equivalent]

```
virtual bool equivalent(int code,
  const ::std::error_condition& condition) const noexcept override;
```

1 *Returns:* True if `condition.category() == *this` and the implementation-defined error code value `code` equates to `static_cast<io2d_error>(condition.value())`. [*Note:* Because of the variations in rendering and presentation technologies available for use on different platforms, the issue of equivalence between error codes and error conditions is one that must be determined by implementors. — *end note*]

```
virtual bool equivalent(const ::std::error_code& ec,
  int condition) const noexcept override;
```

2 *Returns:* True if `ec.category() == *this` and the implementation-defined error code value in `ec.value` equates to `static_cast<io2d_error>(condition)`. [*Note:* Because of the variations in rendering and presentation technologies available for use on different platforms, the issue of equivalence between error codes and error conditions is one that must be determined by implementors. — *end note*]

22.4 `io2d_error_category` non-member functions [io2d.io2derrcat.nonmembers]

```
const ::std::error_category& io2d_category() noexcept;
```

1 *Returns:* A reference to an object of a type derived from `error_category`. All calls to this function shall return references to the same object.

2 *Remarks:* The object's `default_error_condition` virtual function shall behave as specified for the class `error_category`. The object's `message` and `equivalent` virtual functions shall behave as specified for the class `io2d_error_category`. The object's `name` virtual function shall return a pointer to the string `"io2d"`.

23 Class `path` [io2d.path]

23.1 `path` Description [io2d.path.intro]

1 The `path` class is an opaque resource container. It represents one or more geometric paths, which are created using the functionality of the `path_factory` class (24).

2 When a `path` object is set on a `surface` object using `surface::set_path`, the geometric paths represented by it can be stroked or filled.

23.2 `path` synopsis [io2d.path.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class path {
    public:
    // See 1.6.1
    typedef implementation-defined native_handle_type; // Exposition only.
    native_handle_type native_handle(); // Exposition only.

    // 23.3, construct/copy/destroy:
    path() = delete;
    path(const path_factory& pb);
    path(const path& other);
    path& operator=(const path& other);
    path(path&& other);
    path& operator=(path&& other);

    // 23.4, observers:
    ::std::vector<path_data> get_data() const;
    const ::std::vector<path_data>& get_data_ref() const;
    rectangle get_path_extents() const;
  };
} } } } // namespaces std::experimental::io2d::v1
```

23.3 `path` constructors and assignment operators [io2d.path.cons]

```
path(const path_factory& pb);
```

1 *Effects:* Constructs an object of class `path` from a `path_factory` object's observable state.

23.4 `path` observers [io2d.path.observers]

23.4.1 `path::get_path_extents` [io2d.path::get_path_extents]

```
rectangle get_path_extents() const;
```

1 *Returns:* The smallest `rectangle` that can contain the points contained in the `path_data` of `*this`.

2 *Remarks:* When determining the return value of this function, each point should be transformed in the way described in 29.4.23.

23.4.2 `path::get_data` [io2d.path::get_data]

```
::std::vector<path_data> get_data() const;
```

1 *Returns:* A copy of the `vector<path_data>` stored by `*this`.

23.4.3 `path::get_data_ref` [io2d.path::get__data__ref]

```
const ::std::vector<path_data>& get_data_ref() const;
```

1 *Returns:* A const reference to the `vector<path_data>` stored by `*this`.

24 Class `path_factory` [io2d.pathfactory]

24.1 `path_factory` Description [io2d.pathfactory.intro]

1 The `path_factory` class is a factory class that produces `path` objects. The `path` objects produced by a `path_factory` instance are immutable such that subsequent changes to the `path_factory` do not modify a previous `path` object obtained from it.

24.2 `path_factory` synopsis [io2d.pathfactory.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class path_factory {
    // 24.3, construct/copy/destroy:
    path_factory();
    path_factory(const path_factory& x);
    path_factory& operator=(const path_factory& x);
    path_factory(path_factory&& x);
    path_factory& operator=(path_factory&& x);

    // 24.4, modifiers:
    void append(const path& p);
    void append(const path_factory& p);
    void append(const ::std::vector<path_data>& p);
    void reset();
    void new_sub_path();
    void close_path();
    void arc(const point& center, double radius, double angle1,
    double angle2);
    void arc_negative(const point& center, double radius,
    double angle1, double angle2);
    void curve_to(const point& pt0, const point& pt1,
    const point& pt2);
    void line_to(const point& pt);
    void move_to(const point& pt);
    void rect(const rectangle& r);
    void rel_curve_to(const point& dpt0, const point& dpt1,
    const point& dpt2);
    void rel_line_to(const point& dpt);
    void rel_move_to(const point& dpt);
    void set_transform_matrix(const matrix_2d& m);
    void set_origin(const point& pt);

    // 24.5, observers:
    matrix_2d get_transform_matrix() const;
    point get_origin() const;
    bool has_current_point() const;
    point get_current_point() const;
    path get_path() const;
    rectangle get_path_extents() const;
    ::std::vector<path_data> get_data() const;
    const ::std::vector<path_data>& get_data_ref() const;
  };
```

```
} } } } // namespaces std::experimental::io2d::v1
```

24.3 `path_factory` constructors and assignment operators [io2d.pathfactory.cons]

```
path_factory();
```

1 *Effects:* Constructs an object of type `path_factory`.

2 *Postconditions:* `get_data_ref()` returns a const reference to an empty vector. `get_data()` returns an empty vector. `has_current_point()` returns false. `get_origin()` returns a value equivalent to `point{0.0, 0.0}`. `get_transform_matrix()` returns a value equivalent to the return value of `matrix_2d::init_identity()`. `get_path_extents()` returns a value equivalent to `rectangle{0.0, 0.0, 0.0, 0.0}`.

3 *Complexity:* Constant.

24.4 `path_factory` modifiers [io2d.pathfactory.modifiers]

24.4.1 `path_factory::append` [io2d.pathfactory::append]

```
void append(const path& p);
void append(const path_factory& p);
```

1 *Effects:* Appends the results of `p.get_data()` to the `path_data` stored by `*this`.

2 *Postconditions:* If `p.get_data().empty() == false`, the `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. The values returned by `get_transform_matrix()`, `get_origin()`, `has_current_point()`, and `get_current_point()` are the same as they would be if the `path_factory` member function calls necessary to generate the `path_data` in `p.get_data()` had been called.

3 *Complexity:* Linear in the number of elements in `p.get_data()`.

```
void append(const ::std::vector<path_data>& p);
```

4 *Effects:* The same as `path_factory::append_path(const path& p)`.

5 *Throws:* `system_error` with an error code equivalent to `io2d_error::invalid_path_data` and an error category of type `io2d_error_category` if a `path_data` with a type that is not a member of `path_data_type` is encountered or if a `path_data` with a type of `path_data_type::rel_move_to`, `path_data_type::rel_line_to`, or `path_data_type::rel_curve_to` is encountered when no current point is established.

6 *Postconditions:* If `p.empty() == false`, the `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. The values returned by `get_transform_matrix()`, `get_origin()`, `has_current_point()`, and `get_current_point()` are the same as they would be if the `path_factory` member function calls necessary to generate the `path_data` in `p` had been called.

7 *Complexity:* Linear in the number of elements in `p`.

24.4.2 `path_factory::reset` [io2d.pathfactory::reset]

```
void reset();
```

1 *Effects:* Modify `*this` so that its observable state matches that of a `path_factory` object that was just default constructed.

2 *Postcondition:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`.

3 *Complexity:* Constant.

24.4.3 path_factory::new_sub_path [io2d.pathfactory::new_sub_path]

```
void new_sub_path();
```

1 *Effects:* Append a new path_data object to the path_data stored by *this. The appended object's path_data::type is path_data_type::new_sub_path and its path_data::data has path_data::data::unused set to 0.

2 *Postconditions:* The vector<path_data> returned by get_data_ref will change in the ways documented for vector<T>::push_back(const T&). has_current_point() will return false.

3 *Complexity:* Constant.

24.4.4 path_factory::close_path [io2d.pathfactory::close_path]

```
void close_path();
```

1 *Effects:* If has_current_point() == true, append a new path_data object to the path_data stored by *this. The appended object's path_data::type is path_data_type::close_path and its path_data::data has path_data::data::unused set to 0. If has_current_point() == false, do nothing.

2 *Postconditions:* If has_current_point() == true, the vector<path_data> returned by get_data_ref will change in the ways documented for vector<T>::push_back(const T&) and the value returned by get_current_point() will be the value of path_data::data::move from the most recent path_data appended to the path_data stored by *this with a path_data::type of path_data_type::move_to.

3 *Complexity:* Constant.

24.4.5 path_factory::arc [io2d.pathfactory::arc]

```
void arc(const point& center, double radius, double angle1, double angle2);
```

1 *Effects:* Append a new path_data object to the path_data stored by *this. The appended object's path_data::type is path_data_type::arc and its path_data::data has path_data::data::arc::center set to center, path_data::data::arc::radius set to radius, path_data::data::arc::angle1 set to angle1, and path_data::data::arc::angle2 set to angle2.

2 *Postconditions:* The vector<path_data> returned by get_data_ref will change in the ways documented for vector<T>::push_back(const T&). has_current_point() will return true. get_current_point() will return a point placed at point(center.x + radius, center.y and rotated around center such that it is at angle2 radians.

3 *Complexity:* Constant.

24.4.6 path_factory::arc_negative [io2d.pathfactory::arc_negative]

```
void arc_negative(const point& center, double radius, double angle1, double
angle2);
```

1 *Effects:* Append a new path_data object to the path_data stored by *this. The appended object's path_data::type is path_data_type::arc_negative and its path_data::data has path_data::data::arc::center set to center, path_data::data::arc::radius set to radius, path_data::data::arc::angle1 set to angle1, and path_data::data::arc::angle2 set to angle2.

2 *Postconditions:* The vector<path_data> returned by get_data_ref will change in the ways documented for vector<T>::push_back(const T&). has_current_point() will return true. get_current_point() will return a point placed at point(center.x + radius, center.y and rotated around center such that it is at angle2 radians.

3 *Complexity:* Constant.

24.4.7 `path_factory::curve_to` [io2d.pathfactory::curve_to]

```
void curve_to(const point& pt0, const point& pt1, const point& pt2);
```

1 *Effects:* Append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::curve_to` and its `path_data::data` has `path_data::data::curve::pt1` set to `pt0`, `path_data::data::curve::pt2` set to `pt1`, and `path_data::data::curve::pt3` set to `pt2`.

2 *Postcondition:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point()` will return `true`. `get_current_point()` will return `point(pt3)`.

3 *Complexity:* Constant.

24.4.8 `path_factory::line_to` [io2d.pathfactory::line_to]

```
void line_to(const point& pt);
```

1 *Effects:* If `has_current_point() == true`, append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::line_to` and its `path_data::data` has `path_data::data::line` set to `pt`. Otherwise has the same effect as calling `move_to(pt)`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point()` will return `true`. `get_current_point()` will return `point(pt)`.

3 *Complexity:* Constant.

24.4.9 `path_factory::move_to` [io2d.pathfactory::move_to]

```
void move_to(const point& pt);
```

1 *Effects:* Append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::move_to` and its `path_data::data` has `path_data::data::move` set to `pt`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point()` will return `true`. `get_current_point()` will return `point(pt)`.

3 *Complexity:* Constant.

24.4.10 `path_factory::rect` [io2d.pathfactory::rect]

```
void rect(const rectangle& r);
```

1 *Effects:* Append five new `path_data` objects to the `path_data` stored by `*this`. The first appended object's `path_data::type` is `path_data_type::move_to` and its `path_data::data` has `path_data::data::move.x` set to `r.x` and `path_data::data::move.y` set to `r.y`. The second appended object's `path_data::type` is `path_data_type::rel_line_to` and its `path_data::data` has `path_data::data::line.x` set to `r.width` and `path_data::data::line.y` set to `0.0`. The third appended object's `path_data::type` is `path_data_type::rel_line_to` and its `path_data::data` has `path_data::data::line.x` set to `0.0` and `path_data::data::line.y` set to `r.height`. The fourth appended object's `path_data::type` is `path_data_type::rel_line_to` and its `path_data::data` has `path_data::data::line.x` set to `-r.width` and `path_data::data::line.y` set to `0.0`. The fifth appended object's `path_data::type` is `path_data_type::close_path` and its `path_data::data` has `path_data::data::unused` set to `0`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point` will return `false`. `has_current_point()` will return `true`. `get_current_point()` will return `point{r.x, r.y}`.

3 *Complexity:* Constant.

24.4.11 `path_factory::rel_curve_to` [io2d.pathfactory::rel_curve_to]

```
void rel_curve_to(const point& dpt0, const point& dpt1, const point& dpt2);
```

1 *Effects:* Append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::rel_curve_to` and its `path_data::data` has `path_data::data::curve::pt1` set to `pt0`, `path_data::data::curve::pt2` set to `pt1`, and `path_data::data::curve::pt3` set to `pt2`.

2 *Postcondition:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point()` will return `true`. `get_current_point()` will return the result of `get_current_point()` before the function was called with `point(pt3)` added to it.

3 *Complexity:* Constant.

24.4.12 `path_factory::rel_line_to` [io2d.pathfactory::rel_line_to]

```
void rel_line_to(const point& dpt);
```

1 *Effects:* If `has_current_point() == true`, append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::rel_line_to` and its `path_data::data` has `path_data::data::line` set to `pt`. Otherwise has the same effect as calling `move_to(pt)`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point()` will return `true`. `get_current_point()` will return the result of `get_current_point()` before the function was called with `point(pt)` added to it.

3 *Complexity:* Constant.

24.4.13 `path_factory::rel_move_to` [io2d.pathfactory::rel_move_to]

```
void rel_move_to(const point& dpt);
```

1 *Effects:* Append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::rel_move_to` and its `path_data::data` has `path_data::data::move` set to `pt`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`. `has_current_point()` will return `true`. `get_current_point()` will return the result of `get_current_point()` before the function was called with `point(pt)` added to it.

3 *Complexity:* Constant.

24.4.14 `path_factory::set_transform_matrix` [io2d.pathfactory::set_transform_matrix]

```
void set_transform_matrix(const matrix_2d& m);
```

1 *Effects:* Append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::change_matrix` and its `path_data::data` has `path_data::data::matrix` set to `m`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`.

3 *Complexity:* Constant.

24.4.15 `path_factory::set_origin` [io2d.pathfactory::set_origin]

```
void set_origin(const point& pt);
```

1 *Effects:* Append a new `path_data` object to the `path_data` stored by `*this`. The appended object's `path_data::type` is `path_data_type::change_origin` and its `path_data::data` has `path_data::data::origin` set to `pt`.

2 *Postconditions:* The `vector<path_data>` returned by `get_data_ref` will change in the ways documented for `vector<T>::push_back(const T&)`.

3 *Complexity:* Constant.

24.5 `path_factory` observers [io2d.pathfactory.observers]

24.5.1 `path_factory::get_transform_matrix` [io2d.pathfactory::get_transform_matrix]

```
matrix_2d get_transform_matrix() const;
```

1 *Returns:* The value of `path_data::data::matrix` in the element nearest to the end of `get_data_ref()` with a `path_data::type` of `path_data_type::change_matrix` or, if no such element exists, a value equal to the return value of `matrix_2d::init_identity()`.

2 *Complexity:* Constant.

24.5.2 `path_factory::get_origin` [io2d.pathfactory::get_origin]

```
point get_origin() const;
```

1 *Returns:* The value of `path_data::data::origin` in the element nearest to the end of `get_data_ref()` with a `path_data::type` of `path_data_type::change_origin` or, if no such element exists, a value equal to the return value of `point{0.0, 0.0}`.

2 *Complexity:* Constant.

24.5.3 `path_factory::has_current_point` [io2d.pathfactory::has_current_point]

```
bool has_current_point() const;
```

1 *Returns:* True if there is a current point, false if not.

2 *Complexity:* Constant.

24.5.4 `path_factory::get_current_point` [io2d.pathfactory::get_current_point]

```
point get_current_point() const;
```

1 *Returns:* The value of the current point if `has_current_point() == true`, otherwise an undefined but valid value.

2 *Complexity:* Constant.

24.5.5 `path_factory::get_path` [io2d.pathfactory::get_path]

```
path get_path() const;
```

1 *Returns:* `path(*this)`.

24.5.6 `path_factory::get_path_extents` [io2d.pathfactory::get_path_extents]

```
rectangle get_path_extents() const;
```

1 *Returns:* The smallest `rectangle` that can contain the points contained in the `path_data` of `*this`.

2 *Remarks:* When determining the return value of this function, each point should be transformed in the way described in 29.4.23.

24.5.7 `path_factory::get_data` [io2d.pathfactory::get_data]

```
::std::vector<path_data> get_data() const;
```

1 *Returns:* A copy of the `vector<path_data>` stored by `*this`.

24.5.8 `path_factory::get_data_ref` [io2d.pathfactory::get_data_ref]

```
const ::std::vector<path_data>& get_data_ref() const;
```

1 *Returns:* A const reference to the `vector<path_data>` stored by `*this`.

25 Class `device` [io2d.device]

25.1 `device` Description [io2d.device.intro]

1 The `device` class provides access to the underlying rendering and presentation technologies, such a graphics devices, contexts, and swap chains.

25.2 `device` synopsis [io2d.device.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class device {
  public:
    // See 1.6.1
    typedef implementation-defined native_handle_type; // Exposition only.
    native_handle_type native_handle() const; // Exposition only.

    device() = delete;
    device(const device&) = delete;
    device& operator=(const device&) = delete;
    device(device&& other);
    device& operator=(device&& other);

    // 25.3, modifiers:
    void flush();
    void lock();
    void unlock();
  };
} } } } // namespaces std::experimental::io2d::v1
```

25.3 `device` modifiers [io2d.device.modifiers]

25.3.1 `device::flush` [io2d.device::flush]

```
void flush();
```

1 *Effects:* The user will be able to manipulate the underlying rendering and presentation technologies used by the implementation without introducing a race condition. This means that any pending device operations will be executed, batched, or otherwise committed to the underlying technologies. Saved device state, if any, shall also be restored before this function returns.

2 *Remarks:* This function exists primarily to allow the user to take control of the underlying rendering and presentation technologies using an implementation-provided native handle. The implementation's responsibility is to ensure that the user can safely make changes to the state of those underlying technologies. The implementation is not required to ensure that every last operation has fully completed. [*Note:* This function flushes the native device or context that the implementation is using to render to a surface. It is not required to flush the surface. — *end note*]

3 *Notes:* One reason a user might call this function would be because they wished to render UI controls that this library does not provide. As such, the user needs to know that using the underlying rendering technology will not introduce any race conditions. This function, in combination with locking the device, exists to provide that surety. If the underlying technologies internally batch operations in a way that allows them to receive and batch further commands without introducing race conditions, the implementation should return as soon as all pending operations have been submitted to the batch queue.

25.3.2 `device::lock` [io2d.device::lock]

```
void lock();
```

1 *Effects:* Produces all effects of `m.lock()` from *BasicLockable*, 30.2.5.2 in C++ 2014. Implementations shall make this function capable of being recursively reentered.

2 *Throws:* implementation-defined. [*Note:* Implementations should avoid throwing exceptions from this function absent a compelling technical reason such as reaching a maximum level of ownership, in which case an exception of type `::std::system_error` shall be thrown. — *end note*]

25.3.3 `device::unlock` [io2d.device::unlock]

1 *Requires:* Meets all requirements of `m.unlock()` from *BasicLockable*, 30.2.5.2 in C++ 2014.

2 *Effects:* Produces all effects of `m.unlock()` from *BasicLockable*, 30.2.5.2 in C++ 2014. The lock on `m` shall not be fully released until `m.unlock` has been called a number of times equal to the number of times `m.lock` was successfully called.

3 *Throws:* Nothing.

26 Class `font_options_factory` [io2d.fontoptionsfactory]

26.1 `font_options_factory` Description [io2d.fontoptionsfactory.intro]

1 The `font_options_factory` class is a factory class that produces `font_options` objects. The `font_options` objects produced by a `font_options_factory` instance are immutable such that subsequent changes to the `font_options_factory` do not modify a previous `font_options` object obtained from it.

26.2 `font_options_factory` synopsis [io2d.fontoptionsfactory.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class font_options_factory {
  public:
    // 26.3, construct/copy/destroy:
    font_options_factory();
    font_options_factory(const font_options_factory&);
    font_options_factory& operator=(const font_options_factory&);
    font_options_factory(font_options_factory&& other);
    font_options_factory& operator=(font_options_factory&& other);

    // 26.4, modifiers:
    void set_antialias(antialias a);
    void set_subpixel_order(subpixel_order so);

    // 26.5, observers:
    font_options get_font_options() const;
    antialias get_antialias() const;
    subpixel_order get_subpixel_order() const;
  };
} } } } // namespaces std::experimental::io2d::v1
```

26.3 `font_options_factory` constructors and assignment operators [io2d.fontoptionsfactory.cons]

```
font_options_factory();
```

1 *Effects:* Constructs an object of type `font_options_factory`.

2 *Postconditions:* `get_antialias() == antialias::default_antialias`. `get_subpixel_order() == subpixel_order::default_subpixel_order`.

3 *Complexity:* Constant.

26.4 `font_options_factory` modifiers [io2d.fontoptionsfactory.modifiers]

26.4.1 `font_options_factory::set_antialias` [io2d.fontoptionsfactory::set_antialias]

```
void set_antialias(antialias a);
```

1 *Effects:* Set the `antialias` stored by `*this` to the value of `a`.

2 *Complexity:* Constant.

26.4.2 `font_options_factory::set_subpixel_order` [io2d.fontoptionsfactory::set_subpixel_order]

```
void set_subpixel_order(subpixel_order so);
```

1 *Effects:* Set the `subpixel_order` stored by `*this` to the value of `so`.

2 *Complexity:* Constant.

26.5 `font_options_factory` observers [io2d.fontoptionsfactory.observers]

26.5.1 `font_options_factory::get_font_options` [io2d.fontoptionsfactory::get_font_options]

```
font_options get_font_options() const;
```

1 *Returns:* A `font_options` object created with the current value of the `antialias` and the current value of the `subpixel_order` stored by `*this`.

26.5.2 `font_options_factory::get_antialias` [io2d.fontoptionsfactory::get_antialias]

```
antialias get_antialias() const;
```

1 *Returns:* The value of the `antialias` stored by `*this`.

2 *Complexity:* Constant.

26.5.3 `font_options_factory::get_subpixel_order` [io2d.fontoptionsfactory::get_subpixel_order]

```
subpixel_order get_subpixel_order() const;
```

1 *Returns:* The value of the `subpixel_order` stored by `*this`.

2 *Complexity:* Constant.

27 Class `font_options` [io2d.fontoptions]

27.1 `font_options` Description [io2d.fontoptions.intro]

1 The `font_options` class is an opaque resource container. It allows for tweaking how fonts are rendered. It is created using the functionality of the `font_options_factory` class (26).

27.2 `font_options` synopsis [io2d.fontoptions.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class font_options {
    public:
    // See 1.6.1
    typedef implementation-defined native_handle_type; // Exposition only.
    native_handle_type native_handle(); // Exposition only.

    // 27.3, construct/copy/destroy:
    font_options() = delete;
    font_options(const font_options&) = default;
    font_options& operator=(const font_options&) = default;
    font_options(font_options&& other);
    font_options& operator=(font_options&& other);
    font_options(const font_options_factory& factory);

    // 27.4, observers:
    antialias get_antialias() const;
    subpixel_order get_subpixel_order() const;
  };
} } } } // namespaces std::experimental::io2d::v1
```

27.3 `font_options` constructors and assignment operators [io2d.fontoptions.cons]

```
font_options(const font_options_factory& factory);
```

1 *Effects:* Constructs an object of type `font_options` from the observable state of `factory`. value and a `subpixel_order` value.

27.4 `font_options` observers [io2d.fontoptions.observers]

27.4.1 `font_options::get_antialias` [io2d.fontoptions::get_antialias]

```
antialias get_antialias() const;
```

1 *Returns:* The `antialias` value stored by `*this`.

27.4.2 `font_options::get_subpixel_order` [io2d.fontoptions::get_subpixel_order]

```
subpixel_order get_subpixel_order() const;
```

1 *Returns:* The `subpixel_order` value stored by `*this`.

28 Class `font_face` [io2d.fontface]

28.1 `font_face` Description [io2d.fontface.intro]

1 The `font_face` class is an opaque resource container. It represents a font face that is used to render text. It is a base class only and is not directly instantiable.

28.2 `font_face` synopsis [io2d.fontface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class font_face {
  public:
    // See 1.6.1
    typedef implementation-defined native_handle_type;
    native_handle_type native_handle() const;

    // 28.3, construct/copy/destroy:
    font_face() = delete;
    font_face(const font_face&);
    font_face& operator=(const font_face&);
    font_face(font_face&& other);
    font_face& operator=(font_face&& other);
    virtual ~font_face();
  };
} } } } // namespaces std::experimental::io2d::v1
```

28.3 constructors and assignment operators [io2d.fontface.cons]

```
virtual ~font_face();
```

1 *Effects:* Destroys an object of class `font_face`.

2 *Remarks:* Does not throw any exceptions.

29 Class `surface` [io2d.surface]

29.1 `surface` Description [io2d.surface.intro]

1 The `surface` class represents a surface on which graphics are rendered.

2 A `surface` object is a stateful, move-only object.

3 Rendering to a `surface` is accomplished by setting the appropriate state, often in the form of state objects such as `path` and `pattern`, and then calling one or more of the render functions to make the changes specified by the state to the surface resource managed by the `surface` object.

4 A `surface` object is usually created by calling the `make_surface` 29.12.1 function.

29.2 `surface` synopsis [io2d.surface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class surface {
  public:
    // See 1.6.1
    typedef implementation-defined native_handle_type;
    native_handle_type native_handle() const;

    // tuple<dashes, offset>
    typedef ::std::tuple<::std::vector<double>, double> dashes;

    // 29.3, constructors, assignment operators, destructors:
    surface() = delete;
    surface(const surface&) = delete;
    surface& operator=(const surface&) = delete;
    surface(surface&& other);
    surface& operator=(surface&& other);
    surface(const surface& other, content content, int width, int height);
    virtual ~surface();

    // 29.4, state modifiers:
    void finish();
    void flush();
    ::std::shared_ptr<device> get_device();
    void mark_dirty();
    void mark_dirty(const rectangle& rect);
    void set_device_offset(const point& offset);
    void write_to_png(const ::std::string& filename);
    image_surface map_to_image();
    image_surface map_to_image(const rectangle& extents);
    void unmap_image(image_surface& image);
    void save();
    void restore();
    void set_pattern();
    void set_pattern(const pattern& source);
    void set_antialias(antialias a);
    void set_dashes();
    void set_dashes(const dashes& d);
    void set_fill_rule(fill_rule fr);
```

```
void set_line_cap(line_cap lc);
void set_line_join(line_join lj);
void set_line_width(double width);
void set_miter_limit(double limit);
void set_compositing_operator(compositing_operator co);
void clip();
void reset_clip();
void set_path();
void set_path(const path& p);

// 29.5, render modifiers:
void fill();
void fill(const surface& s);
void mask(const pattern& pttn);
void mask(const surface& surface);
void mask(const surface& surface, const point& origin);
void paint();
void paint(const surface& s);
void paint(double alpha);
void paint(const surface& s, double alpha);
void stroke();
void stroke(const surface& s);
void show_text(const ::std::string& utf8);
void show_glyphs(const ::std::vector<glyph>& glyphs);
void show_text_glyphs(const ::std::string& utf8,
  const ::std::vector<glyph>& glyphs,
  const ::std::vector<text_cluster>& clusters,
  bool clusterToGlyphsMapReverse = false);

// 29.6, transformation modifiers:
void set_matrix(const matrix_2d& matrix);

// 29.7, font modifiers:
void select_font_face(const ::std::string& family, font_slant slant,
  font_weight weight);
void set_font_size(double size);
void set_font_matrix(const matrix_2d& matrix);
void set_font_options(const font_options& options);
void set_font_face(const font_face& font_face);
void set_scaled_font(const scaled_font& scaled_font);

// 29.8, state observers:
content get_content() const;
point get_device_offset() const;
bool has_surface_resource() const;
pattern get_pattern() const;
antialias get_antialias() const;
int get_dashes_count() const;
dashes get_dashes() const;
fill_rule get_fill_rule() const;
line_cap get_line_cap() const;
line_join get_line_join() const;
double get_line_width() const;
double get_miter_limit() const;
compositing_operator get_compositing_operator() const;
```

```
    double get_tolerance() const;
    rectangle get_clip_extents() const;
    bool in_clip(const point& pt) const;
    ::std::vector<rectangle> get_clip_rectangles() const;

    // 29.9, render observers:
    rectangle get_fill_extents() const;
    bool in_fill(const point& pt) const;
    rectangle get_stroke_extents() const;
    bool in_stroke(const point& pt) const;
    font_extents get_font_extents() const;
    text_extents get_text_extents(const ::std::string& utf8) const;
    text_extents get_glyph_extents(const ::std::vector<glyph>& glyphs) const;

    // 29.10, transformation observers:
    matrix_2d get_matrix() const;
    point user_to_device() const;
    point user_to_device_distance() const;
    point device_to_user() const;
    point device_to_user_distance() const;

    // 29.11, font observers:
    matrix_2d get_font_matrix() const;
    font_options get_font_options() const;
    font_face get_font_face() const;
    scaled_font get_scaled_font() const;
  };

  // 29.12, non-member functions:
  surface make_surface(implementation-defined) implementation-defined;
} } } } // namespaces std::experimental::io2d::v1
```

29.3 `surface` constructors, assignment operators, and destructors [io2d.surface.cons]

```
surface(const surface& other, content content, int width, int height);
```

1 *Effects:* Constructs a `surface` object. It shall, to the extent possible, have the same observable state as `other`. Its content shall be transparent black if it has an alpha channel or black if it does not.

2 *Remarks:* The `surface` object that is constructed takes its initial state from `other` to the extent possible, but is not observably connected to `other` such that changes to either surface, including destruction, do not affect the observable state of the other surface.

```
virtual ~surface();
```

3 *Effects:* Destroys an object of class `surface`.

4 *Remarks:* Does not throw any exceptions.

29.4 `surface` state modifiers [io2d.surface.modifiers.state]

29.4.1 `surface::finish` [io2d.surface::finish]

```
void finish();
```

1 *Effects:* Releases all resources associated with the `surface`.

2 *Remarks:* Once this function has been called, the surface is considered finished. The only valid operations on a finished surface are destruction and calling `finish` again (which will return without

doing anything since the surface is already finished). Any other operation on a finished surface or any attempt to use a finished surface as an argument to a function produces undefined behavior.

29.4.2 `surface::flush` [io2d.surface::flush]

```
void flush();
```

1 *Effects:* The user will be able to use the implementation's underlying surface without introducing a race condition. This means that any pending surface operations will be executed, batched, or otherwise committed to the underlying technologies. Saved surface state, if any, shall also be restored before this function returns.

2 *Remarks:* This function exists primarily to allow the user to take control of the underlying surface using an implementation-provided native handle. The implementation's responsibility is to ensure that the user can safely use the underlying surface. [*Note:* This function flushes the native surface so that it can safely be rendered to or, if the underlying technology supports it, be used as a texture to be sampled from. It is not required to flush the device or context that is rendering to it. — *end note*]

29.4.3 `surface::get_device` [io2d.surface::get_device]

```
::std::shared_ptr<device> get_device();
```

1 *Returns:* A shared pointer to the `device` object for this `surface`.

29.4.4 `surface::mark_dirty` [io2d.surface::mark_dirty]

```
void mark_dirty();
```

1 *Effects:* Informs the implementation that external changes were made to the underlying surface.

2 *Remarks:* A program shall call this function first when using the `surface` object after external changes are made to its underlying surface; no diagnostic is required.

```
void mark_dirty(const rectangle& rect);
```

3 *Effects:* Informs the implementation that external changes were made to the underlying surface within the area denoted by `rect`.

4 *Remarks:* This function shall be at least as efficient as `mark_dirty()`. A program shall call this function first when using the `surface` object after external changes are made to its underlying surface; no diagnostic is required.

29.4.5 `surface::set_device_offset` [io2d.surface::set_device_offset]

```
void set_device_offset(const point& offset);
```

1 *Effects:* Stores the value of `offset`. The value will be added to all coordinates after they have been transformed by the `surface` object's transformation matrix.

29.4.6 `surface::write_to_file` [io2d.surface::write_to_file]

```
void write_to_file(const ::std::string& filename);
```

1 *Effects:* implementation-defined.

2 *Notes:* This functionality is expected to change and evolve to make use of the Filesystem TS in the future. For now implementations should write the surface's data to an image file.

29.4.7 `surface::map_to_image` [io2d.surface::map_to_image]

```
image_surface map_to_image();
```

1 *Effects:* Maps `*this` to a newly created `image_surface` which allows direct manipulation of the underlying surface. Whether changes are committed to the underlying surface immediately or only when the surface is unmapped is implementation-defined. Implementations may specify different commit behaviors for `surface::map_to_image()` and `surface::map_to_image(const rectangle&)`.

2 *Returns:* An `image_surface` which, when changed, will change the underlying surface of `*this`. [*Note:* Use of the returned `image_surface` is very limited. The only valid operations that may be performed with it are move construction, move assignment, destruction, `image_surface::set_data`, `image_surface::get_data`, `image_surface::get_format`, `image_surface::get_width`, `image_surface::get_height`, `image_surface::get_stride`, and passing it as an argument when calling `surface::unmap_image` on the `surface` object that created it. All other operations produce undefined behavior. — *end note*]

3 *Remarks:* A `surface` object shall only be mapped once at any time; no diagnostic is required. A `surface` object that is mapped shall not be used until it is unmapped; no diagnostic is required. A mapped `surface` object can be unmapped by either destroying the `image_surface` object returned by this function or by passing the `image_surface` object returned by this function to the `unmap_image` member function of the `surface` object that created it.

```
image_surface map_to_image(const rectangle& extents);
```

4 *Effects:* Creates an `image_surface` that allows direct manipulation of the underlying surface. The `image_surface` will be scoped to the area of the underlying surface defined by `extents`. Whether changes are committed to the underlying surface immediately or only when the surface is unmapped is implementation-defined. Implementations may specify different commit behaviors for `surface::map_to_image()` and `surface::map_to_image(const rectangle&)`.

5 *Returns:* An `image_surface` which, when changed, will change the underlying surface of `*this`. [*Note:* Use of the returned `image_surface` is very limited. The only valid operations that may be performed with it are move construction, move assignment, destruction, `image_surface::set_data`, `image_surface::get_data`, `image_surface::get_format`, `image_surface::get_width`, `image_surface::get_height`, `image_surface::get_stride`, and passing it as an argument when calling `surface::unmap_image` on the `surface` object that created it. All other operations produce undefined behavior. — *end note*]

6 *Remarks:* A `surface` object shall only be mapped once at any time; no diagnostic is required. A `surface` object that is mapped shall not be used until it is unmapped; no diagnostic is required. A mapped `surface` object can be unmapped by either destroying the `image_surface` object returned by this function or by passing the `image_surface` object returned by this function to the `unmap_image` member function of the `surface` object that created it.

29.4.8 `surface::unmap_image` [io2d.surface::unmap_image]

```
void unmap_image(image_surface& image);
```

1 *Requires:* `image` must have been created by calling `surface::map_to_image` on `*this`.

2 *Effects:* Unmaps `*this` and commits any uncommitted changes that were made to the underlying surface using `image`.

3 *Throws:* `::std::invalid_argument` if `image` was not created by calling `surface::map_to_image` on `*this`.

4 *Remarks:* Except in move construction, move assignment, and destruction, any use of `image` after it has been passed as an argument to this function produces undefined behavior.

29.4.9 `surface::save` [io2d.surface::save]

```
void save();
```

1 *Effects:* Saves the current observable state of `*this` to a newly created entry on the internal stack of saved states.

29.4.10 `surface::restore` [io2d.surface::restore]

```
void restore();
```

1 *Requires:* A matching call to `surface::save` on `*this`.

2 *Effects:* Restores the observable state of `*this` from the most recent entry on the internal stack of saved states and removes that entry from the stack.

3 *Throws:* `::std::system_error` with an error category of `io2d_error_category` and an error condition of `io2d_error::invalid_restore` if there are no entries on the stack of saved states when this function is called.

29.4.11 `surface::set_pattern` [io2d.surface::set_pattern]

```
void set_pattern();
```

1 *Effects:* Unsets the current `pattern` object and sets a default `pattern` object. [*Note:* The default `pattern` object is always an opaque black solid color pattern, equivalent to the pattern created by `solid_color_pattern_factory(rgba_color::black).get_pattern()`. It is not required to be the same object throughout the lifetime of the program. — *end note*]

2 *Postconditions:* `get_pattern()` returns a default `pattern` object.

```
void set_pattern(const pattern& source);
```

3 *Effects:* Unsets the current `pattern` object and sets `source` as the current `pattern` object.

4 *Postconditions:* `get_pattern()` returns a `pattern` object with the same observable state as `source` object.

29.4.12 `surface::set_antialias` [io2d.surface::set_antialias]

```
void set_antialias(antialias a);
```

1 *Effects:* Sets the current `antialias` state to `a`.

2 *Postconditions:* `get_antialias()` returns the value of `a`.

29.4.13 `surface::` [io2d.surface::]

```
void set_dashes();
```

1 *Effects:* Sets the current dash pattern to be a solid line.

2 *Postconditions:* `get_dashes()` will return the equivalent of `::std::make_tuple::std::<vector<double>, double>({ }, 0.0)`. `get_dashes_count()` will return `0`.

```
void set_dashes(const dashes& d);
```

3 *Effects:* Sets the current dash pattern to the value of `d`.

4 *Postconditions:* `get_dashes()` will return the equivalent of `d`. `get_dashes_count()` will return the value of `static_cast<int>(::std::get<0>(d).size())`.

29.4.14 surface::set_fill_rule [io2d.surface::set_fill_rule]

```
void set_fill_rule(fill_rule fr);
```

1 *Effects:* Sets the current fill rule to the value of `fr`.

2 *Postconditions:* `get_fill_rule()` will return the value of `fr`.

29.4.15 surface::set_line_cap [io2d.surface::set_line_cap]

```
void set_line_cap(line_cap lc);
```

1 *Effects:* Sets the current line cap to the value of `lc`.

2 *Postconditions:* `get_line_cap()` will return the value of `lc`.

29.4.16 surface::set_line_join [io2d.surface::set_line_join]

```
void set_line_join(line_join lj);
```

1 *Effects:* Sets the current line join to the value of `lj`.

2 *Postconditions:* `get_line_join()` will return the value of `lj`.

29.4.17 surface::set_line_width [io2d.surface::set_line_width]

```
void set_line_width(double width);
```

1 *Effects:* Sets the current line width to `::std::max(width, 0.0)`.

2 *Postconditions:* `get_line_width()` will return `::std::max(width, 0.0)`.

3 *Remarks:* Line width is in pre-transformation units and is used with the `surface::stroke` functions. It postulates a round brush with a diameter of the current line width. The shape and size of the stroke when rendered depends on other state data. See `surface::stroke`29.5.1 for more information.

29.4.18 surface::set_miter_limit [io2d.surface::set_miter_limit]

```
void set_miter_limit(double limit);
```

1 *Effects:* Sets the current miter limit to the value of `::std::max(limit, 1.0)`. This limit only applies when the current line join is set to `line_join::miter_or_bevel`.

2 *Postconditions:* `get_miter_limit` will return `::std::max(limit, 1.0)`.

3 *Remarks:* The miter limit works as follow. The length of the miter is divided by the width of the line. If the resulting value is greater than `limit`, the line join will be beveled, otherwise it will be mitered. [*Note:* When the join angle is less than 0.0005 radians, it is implementation-defined whether a mitered join or a beveled joins will be produced. Specifically, when the join angle is less than 0.0005 radians, implementations may switch to beveled rendering instead of mitered rendering at any arbitrary *implementation-defined*angle value, provided that the value is less than 0.0005 radians and that all joins with angles below the *implementation-defined*angle value are rendered as beveled. — *end note*]

29.4.19 surface::set_compositing_operator [io2d.surface::set_compositing_operator]

```
void set_compositing_operator(compositing_operator co);
```

1 *Effects:* Sets the current compositing operator to the value of `co`.

2 *Postconditions:* `get_compositing_operator` will return the value of `co`.

29.4.20 `surface::clip` [io2d.surface::clip]

```
void clip();
```

1 *Effects:* Sets the current clip area to be the intersection of the current clip area and the current path where the current path's area is determined in the same way as if the current path were filled according to the current fill rule.

2 *Notes:* The clip area never increases as a result of this function.

29.4.21 `surface::reset_clip` [io2d.surface::reset_clip]

```
void reset_clip();
```

1 *Effects:* Resets the current clip area to be an unbounded, infinitely large area. All points are within the resulting clip area.

29.4.22 `surface::set_path` [io2d.surface::set_path]

```
void set_path();
```

1 *Effects:* Set's the current `path` to be an empty path.

2 *Notes:* The empty `path` object is not supplied by the user. The implementation is expected to provide an empty `path` object. Since a `path` object is immutable, an implementation should create an empty path object in advance for maximum efficiency.

```
void set_path(const path& p);
```

3 *Effects:* Set's the current `path` to `p`.

4 *Remarks:* Processing the path data so that it is properly transformed can be done at the time it is first set as a path on a `surface` object or any time before that. The untransformed `vector<path_data>` must be retained to ensure that a path can be properly recreated at any time. The steps for converting an untransformed `vector<path_data>` to transformed path data are found at 29.4.23.

29.4.23 `vector<path_data>` transformation steps [io2d.surface.pathtransform]

1 [*Note:* The "as if" rule applies here. For purposes of exposition, it is assumed that there is a native geometry object which supports creating a line from its current point to a supplied point and creating a cubic Bézier curve using its current point as the start point (with the control points and end point supplied by the user), a close path instruction, and which tracks its current point and its last move to point. In reality the native geometry object may provide more (or less) capabilities. It is possible to render with as little native support as the ability to stroke a line and fill a rectangular area. — *end note*]

2

1. Create a `matrix_2d m` and initialize it to `matrix_2d::init_identity();`.
2. Create a `point origin` and initialize it to `{ 0.0, 0.0 }`.
3. Create a `point currentPoint`.
4. Create a `bool hasCurrentPoint` and initialize it to `false`.
5. Create a `point lastMovePoint`.
6. Create a `bool hasLastMovePoint` and initialize it to `false`.
7. Create the following variable: `const auto& pathData = p.get_data_ref();`.

8. Create the following loop: `for (const auto& item :  pathData) { }`

9. Within the body of the loop, evaluate `item.type`:
   a) If `item.type` is `path_data_type::move_to` perform the following actions. Set `currentPoint` equal to the value of `item.data.move`. Set `hasCurrentPoint` equal to `true`. Set the native geometry's current point equal to the value of `m.transform_point(item.data.move - origin) + origin`. Set `lastMovePoint` equal to the value of `item.data.move`. Set `hasLastMovePoint` equal to `true`. Set the native geometry's last move to point equal to the value of `m.transform_point(item.data.move - origin) + origin`.
   b) If `item.type` is `path_data_type::line_to` perform the following actions. Set `currentPoint` equal to the value of `item.data.line`. If `hasCurrentPoint` is true, instruct the native geometry to create a line to `m.transform_point(item.data.line - origin) + origin`. Regardless, set `hasCurrentPoint` to true and set the native geometry's current point equal to `m.transform_point(item.data.line - origin) + origin`.
   c) If `item.type` is `path_data_type::curve_to` perform the following actions. If `hasCurrentPoint` is equal to `false`, set the native geometry's current point equal to `m.transform_point(item.data.curve.pt1 - origin) + origin`. Regardless, instruct the native geometry to create a cubic Bézier curve using `m.transform_point(item.data.curve.pt1 - origin) + origin` as the first control point, `m.transform_point(item.data.curve.pt2 - origin) + origin` as the second control point, and `m.transform_point(item.data.curve.pt3 - origin) + origin` as the end point. Set the native geometry's current point to `m.transform_point(item.data.curve.pt3 - origin) + origin`. Set `currentPoint` equal to the value of `item.data.curve.pt3 + currentPoint`. Set `hasCurrentPoint` to `true`.
   d) If `item.type` is `path_data_type::new_sub_path` perform the following actions. Set `hasCurrentPoint` to `false`. Set `hasMoveToPoint` to false.
   e) If `item.type` is `path_data_type::close_path` perform the following actions.

(2.1) — If `hasCurrentPoint` is `true` when this `item.type` is encountered, instruct the native geometry to create a line connecting its current point to its last move to point, with the connection at the last move to point being a join instead of being two separate segment end points. Set `currentPoint` to `lastMoveToPoint`. Instruct the native geometry to set its current point to be equal to its last move to point.

(2.2) — If `hasCurrentPoint` is `false` when this `item.type` is encountered, do nothing.

   f) If `item.type` is `path_data_type::rel_move_to` perform the following actions. Set `currentPoint` equal to `currentPoint` plus the value of `item.data.move`. Set `hasCurrentPoint` equal to `true`. Set the native geometry's current point equal to the value of `m.transform_point(currentPoint - origin) + origin`. Set `lastMovePoint` equal to the value of `currentPoint`. Set `hasLastMovePoint` equal to `true`. Set the native geometry's last move to point equal to the value of `m.transform_point(currentPoint - origin) + origin`.
   g) If `item.type` is `path_data_type::rel_line_to` perform the following actions. Set `currentPoint` equal to the value of `item.data.line + currentPoint`. If `hasCurrentPoint` is true, instruct the native geometry to create a line to `m.transform_point(currentPoint - origin) + origin`. Regardless, set `hasCurrentPoint` to true and set the native geometry's current point equal to `m.transform_point(currentPoint - origin) + origin`.
   h) If `item.type` is `path_data_type::rel_curve_to` perform the following actions. If `hasCurrentPoint` is equal to `false`, set the native geometry's current point equal to `m.transform_point((item.data.curve.pt1 + currentPoint) - origin) + origin`. Regardless, instruct the native geometry to create a cubic Bézier curve using `m.transform_point((item.data.curve.pt1 + currentPoint) - origin) + origin` as the first control point, `m.transform_point((item.data.curve.pt2 +`

currentPoint) - origin) + origin as the second control point, and m.transform_point((item.data.curve.p + currentPoint) - origin) + origin as the end point. Set the native geometry's current point to m.transform_point((item.data.curve.pt3 + currentPoint) - origin) + origin. Set hasCurrentPoint to true. Set currentPoint equal to the value of item.data.curve.pt3 + currentPoint. Set hasCurrentPoint to true.

i) If item.type is path_data_type::arc perform the following actions. Convert the arc into one or more cubic Bézier curves that, when rendered in sequence, approximate the arc as closely as reasonably possible.

(2.3) — If hasCurrentPoint is true, create a line from the current point to the starting point of the first curve in the manner prescribed when item.type is path_data_type::line_to.

(2.4) — Otherwise move to the starting point of the first curve in the manner prescribed when item.type is path_data_type::move_to.

Process each curve in the manner prescribed when item.type is path_data_type::curve_to. [*Note:* Done properly, this results in an arc that is properly transformed with due consideration of the current origin such that everything from circles to rotated elliptical arcs are possible. It also leaves the current point and the last move to point set to their appropriate values. — *end note*]

j) If item.type is path_data_type::arc_negative perform the following actions. Convert the arc into one or more cubic Bézier curves that, when rendered in sequence, approximate the arc as closely as reasonably possible.

(2.5) — If hasCurrentPoint is true, create a line from the current point to the starting point of the first curve in the manner prescribed when item.type is path_data_type::line_to.

(2.6) — Otherwise move to the starting point of the first curve in the manner prescribed when item.type is path_data_type::move_to.

Process each curve in the manner prescribed when item.type is path_data_type::curve_to. [*Note:* Done properly, this results in an arc that is properly transformed with due consideration of the current origin such that everything from circles to rotated elliptical arcs are possible. It also leaves the current point and the last move to point set to their appropriate values. — *end note*]

k) If item.type is path_data_type::change_matrix perform the following actions. Set m equal to the value of item.data.matrix.

l) If item.type is path_data_type::change_origin perform the following actions. Set origin equal to the value of item.data.origin.

m) If item.type is any other value, throw a system_error with error condition io2d_error::invalid_path_data.

29.5 surface render modifiers [io2d.surface.modifiers.render]

29.5.1 surface::stroke [io2d.surface::stroke]

```
void stroke();
void stroke(const surface& s);
```

1 *Effects:...*

2 *Postconditions:...*

3 *Complexity:...*

```
surface make_surface(implementation-defined) implementation-defined;
```

1 *Returns:* A `surface` object. The default initial state of the returned `surface` object is listed in Table 19. [*Note:* Implementations may return a `surface` object with a different initial state. See the Remarks section below. — *end note*]

Table 19 — `make_surface` Default Initial State Values

Observer	Return Value
a	b

2 *Remarks:* The name, return type, and default initial values for the observable states of the returned `surface` object are the only semantic elements specified for `make_surface`. The default initial values for the observable state of the returned `surface` object are given in Table 19. Implementations are free to provide different initial values provided that the differences are documented.

Index

Index of library names

Index of implementation-defined behavior

The entries in this section are rough descriptions; exact specifications are at the indicated page in the general text.