Document Number: P0267R5
Date: 2017-06-19
Revises: P0267R4
Reply to: Michael B. McLaughlin
mikebmcl@gmail.com

Herb Sutter
Microsoft Inc.
hsutter@microsoft.com

Jason Zink
jzink_1@yahoo.com

Guy Davidson
guy@creative-assembly.com
Audience: LEWG

A Proposal to Add 2D Graphics Rendering and Display to C++

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 Scope [io2d.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 2D computer graphics.

2 Normative references [io2d.refs]

1 The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 2382 (all parts), *Information technology — Vocabulary*

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

(1.4) — ISO/IEC 10918-1, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines*

(1.5) — ISO 12639, *Graphic technology – Prepress digital data exchange – Tag image file format for image technology (TIFF/IT)*

(1.6) — ISO/IEC 15948 *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG) Functional specification*

(1.7) — 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*

(1.8) — ISO 15076-1, *Image technology colour management — Architecture, profile format and data structure — Part 1: Based on ICC.1:2004-10*

(1.9) — IEC 61966-2-1, *Colour Measurement and Management in Multimedia Systems and Equipment - Part 2-1: Default RGB Colour Space - sRGB*

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

(1.11) — ISO 80000-2:2009, *Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology*

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

2 The compressed image data format described in ISO/IEC 10918-1 is hereinafter called the *JPEG format*.

3 The tag image file format described in ISO 12639 is hereinafter called the *TIFF format*. The datastream and associated file format described in ISO/IEC 15948 is hereinafter called the *PNG format*.

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

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

3 Terms and definitions [io2d.defns]

For the purposes of this document, the following terms and definitions apply. ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

1 Terms that are used only in a small portion of this document are defined where they are used and italicized where they are defined.

3.1 [io2d.defns.stndcrdspace]

standard coordinate space
a Euclidean plane described by a Cartesian coordinate system where the first coordinate is measured along a horizontal axis, called the x axis, oriented from left to right, the second coordinate is measured along a vertical axis, called the y axis, oriented from top to bottom, and rotation of a point around the origin by a positive value expressed in radians is counterclockwise

3.2 [io2d.defns.visdata]

visual data
data representing color, transparency, or some combination thereof

3.3 [io2d.defs.channel]

channel
a component of visual data with a defined bit size

3.4 [io2d.defns.visdatafmt]

visual data format
a specification that defines a total bit size, a set of one or more channels, and each channel's role, bit size, and location relative to the upper (high-order) bit

3.5 [io2d.defns.visdataelem]

visual data element
an item of visual data with a defined visual data format

3.6 [io2d.defns.alpha]

alpha
visual data representing transparency

3.7 [io2d.defns.pixel]

pixel
a discrete, rectangular visual data element

3.8 [io2d.defns.alias]

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

3.9 [io2d.defns.artifact]

artifact
an error in the results of the application of a composing operation

3.10 **[io2d.defns.antialias]**

anti-aliasing

the application of a function or algorithm while composing to reduce aliasing [*Note:* Certain algorithms can produce "better" results, i.e. results with less artifacts or with less pronounced artifacts, when rendering text with anti-aliasing due to the nature of text rendering. As such, it often makes sense to provide the ability to choose one type of anti-aliasing for text rendering and another for all other rendering and to provide different sets of anti-aliasing types to choose from for each of the two operations. — *end note*]

3.11 **[io2d.defns.aspectratio]**

aspect ratio

the ratio of the width to the height of a rectangular area

3.12 **[io2d.defns.additivecolor]**

additive color

a color defined by the emissive intensity of its color channels

3.13 **[io2d.defns.colormodel]**

color model

an ideal, mathematical representation of colors which often uses color channels

3.14 **[io2d.defns.rgbcolormodel]**

RGB color model

⟨RGB⟩ additive color model using red, green, and blue color channels

3.15 **[io2d.defns.rgbacolormodel]**

RGBA color model

⟨RGBA⟩ RGB color model with an alpha channel

3.16 **[io2d.defns.colorspace]**

color space

unambiguous mapping of values to colorimetric colors

3.17 **[io2d.defns.srgbcolorspace]**

sRGB color space

⟨sRGB⟩ additive color space defined in IEC 61966-2-1 that is based on the RGB color model

3.18 **[io2d.defns.bezier.cubic]**

Bézier curve

⟨cubic⟩ curve defined by the equation $f(t) = (1-t)^3 \times P_0 + 3 \times t \times (1-t)^2 \times P_1 + 3 \times t^2 \times (1-t) \times P_2 + t^3 \times t \times P_3$ where t is in the range `[0, 1]`, P_0 is the starting point, P_1 is the first control point, P_2 is the second control point, and P_3 is the ending point

3.19 **[io2d.defns.bezier.quadratic]**

Bézier curve

⟨quadratic⟩ curve defined by the equation $f(t) = (1-t)^2 \times P_0 + 2 \times t \times (1-t) \times P_1 + t^2 \times t \times P_2$ where t is in the range `[0, 1]`, P_0 is the starting point, P_1 is the control point, and P_2 is end point

3.20 **[io2d.defns.filter]**

filter

mathematical function that determines the visual data value of a point for a graphics data graphics resource

3.21 **[io2d.defns.graphicsdata]**
graphics data
⟨graphics data⟩ visual data stored in an unspecified form

3.22 **[io2d.defns.graphics.raster]**
graphics data
⟨raster graphics data⟩ visual data stored as pixels that is accessible as-if it was an array of rows of pixels beginning with the pixel at the integral point $(0, 0)$

3.23 **[io2d.defns.graphicsresource]**
graphics resource
⟨graphics resource⟩ object of unspecified type used by an implementation [*Note:* By its definition a graphics resource is an implementation detail. Often it will be a graphics subsystem object (e.g. a graphics device or a render target) or an aggregate composed of multiple graphics subsystem objects. However the only requirement placed upon a graphics resource is that the implementation is able to use it to provide the functionality required of the graphics resource. — *end note*]

3.24 **[io2d.defns.graphicsresource.graphicsdata]**
graphics resource
⟨graphics data graphics resource⟩ object of unspecified type used by an implementation to provide access to and allow manipulation of visual data

3.25 **[io2d.defns.pixmap]**
pixmap
raster graphics data graphics resource

3.26 **[io2d.defns.point]**
point
⟨point⟩ coordinate designated by a floating point x axis value and a floating point y axis value within the standard coordinate space

3.27 **[io2d.defns.point.integral]**
point
⟨integral point⟩ coordinate designated by an integral x axis value and an integral y axis value within the standard coordinate space

3.28 **[io2d.defns.premultipliedformat]**
premultiplied format
format with color and alpha where each color channel is normalized and then multiplied by the normalized alpha channel value [*Example:* Given the 32-bit non-premultiplied RGBA pixel with 8 bits per channel {255, 0, 0, 127} (half-transparent red), when normalized it would become {1.0f, 0.0f, 0.0f, 0.5f}. As such, in premultiplied, normalized format it would become {0.5f, 0.0f, 0.0f, 0.5f} as a result of multiplying each of the three color channels by the alpha channel value. — *end example*]

3.29 **[io2d.defns.graphicsstatedata]**
graphics state data
data which specify how some part of the process of rendering or of a composing operation shall be performed in part or in whole

3.30 **[io2d.defns.graphicssubsystem]**
graphics subsystem
collection of unspecified operating system and library functionality used to render and display 2D computer graphics

3.31 [io2d.defns.normalize]
normalize
to map a closed set of evenly spaced values in the range $[0, x]$ to an evenly spaced sequence of floating point values in the range $[0, 1]$ [*Note:* The definition of normalize given is the definition for normalizing unsigned input. Signed normalization, i.e. the mapping of a closed set of evenly spaced values in the range $[-x, x)$ to an evenly spaced sequence of floating point values in the range $[-1, 1]$, also exists but is not used in this Technical Specification. — *end note*]

3.32 [io2d.defns.render]
render
to transform a path group into graphics data in the manner specified by a set of graphics state data

3.33 [io2d.defns.renderingoperation]
rendering operation
an operation that performs rendering

3.34 [io2d.defns.compose]
compose
to combine part or all of a source graphics data graphics resource with a destination graphics data graphics resource in the manner specified by a composition algorithm

3.35 [io2d.defns.composingoperation]
composing operation
an operation that performs composing

3.36 [io2d.defns.compositionalgorithm]
composition algorithm
an algorithm that combines a source visual data element and a destination visual data element producing a visual data element that has the same visual data format as the destination visual data element

3.37 [io2d.defns.renderingandcomposingop]
rendering and composing operation
an operation that is either a composing operation or a rendering operation followed by a composing operation

3.38 [io2d.defns.sample]
sample
to use a filter to obtain the visual data for a given point from a graphics data graphics resource

3.39 [io2d.defns.colorstop]
color stop
a tuple composed of a floating point offset value in the range $[0, 1]$ and a color value

3.40 [io2d.defns.pathseg]
path segment
line, Bézier curve, or arc, each of which has a start point and an end point

3.41 [io2d.defns.controlpt]
control point
point other than the start point and end point that is used in defining a Bézier curve

3.42 [io2d.defns.degenepathseg]
degenerate path segment
path segment that has the same values for its start point, end point, and, if any, control points

3.43 [io2d.defns.initialpathseg]

initial path segment

path segment whose start point is not defined as being the end point of another path segment [*Note:* It is possible for the initial path segment and final path segment to be the same path segment. — *end note*]

3.44 [io2d.defns.finalpathseg]

final path segment

path segment whose end point does not define the start point of any other path segment [*Note:* It is possible for the initial path segment and final path segment to be the same path segment. — *end note*]

3.45 [io2d.defns.newpathinstruction]

path instruction

⟨new path instruction⟩ instruction that creates a new path

3.46 [io2d.defns.closepathinstruction]

path instruction

⟨close path instruction⟩ instruction that creates a line path segment from the current point to the , a path and establishes a new path

3.47 [io2d.defns.pathitem]

path item

path segment, new path instruction, close path instruction, or path group instruction

3.48 [io2d.defns.path]

path

collection of path items where the end point of each path segment, except the final path segment, defines the start point of exactly one other path segment in the collection

3.49 [io2d.defns.degenpath]

degenerate path

path composed entirely of a new path instruction, zero or more degenerate path segments, zero of more path group items, and, optionally, a close path instruction

3.50 [io2d.defns.currentpoint]

current point

point used as the start point of a path segment

3.51 [io2d.defns.newpathpt]

new path point

point in a path that is the start point of the initial path segment

3.52 [io2d.defns.pathgroup]

path group

collection of paths

3.53 [io2d.defns.pathgrptransform]

path group transformation matrix

affine transformation matrix used to apply affine transformations to the points in a path group

3.54 [i]

path group instruction

nstruction that modifies the path group transformation matrix

3.55 [io2d.defns.closedpath]

closed path

path with one or more path segments where the new path point is used to define the end point of the path's final path segment

3.56 [io2d.defns.openpath]

open path

path with one or more path segments where the new path point is not used to define the end point of the path's final path segment [*Note:* Even if the start point of the initial path segment and the end point of the final path segment are assigned the same coordinates, the path is still an open path since the final path segment's end point is not defined as being the start point of the initial segment but instead merely happens to have the same value as that point. — *end note*]

4 Error reporting [io2d.err.report]

1 2D graphics library functions that can produce errors occasionally provide two overloads: one that throws an exception to report errors and another that reports errors using an `error_code` object. This provides for situations where errors are not truly exceptional.

2 report errors as follows, unless otherwise specified:

3 When an error prevents the function from meeting its specifications:

(3.1) — Functions that do not take argument of type `error_code&` throw an exception of type `system_error` or of a `system_error`-derived type. The exception object shall include the enumerator specified by the function as part of its observable state.

(3.2) — Functions that take an argument of type `error_code&` assigns the specified enumerator to the provided `error_code` object and then returns.

4 Failure to allocate storage is reported by throwing an exception as described in [res.on.exception.handling] in N4618.

5 Destructor operations defined in this Technical Specification shall not throw exceptions. Every destructor in this Technical Specification shall behave as if it had a non-throwing exception specification.

6 If no error occurs in a function that takes an argument of type `error_code&`, `error_code::clear` shall be called on the `error_code` object immediately before the function returns.

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

```
namespace std { namespace experimental {
  namespace io2d { inline namespace v1 {

  using dashes = tuple<vector<float>, float>;

  enum class fill_rule;
  enum class line_cap;
  enum class line_join;
  enum class compositing_op;
  enum class format;
  enum class wrap_mode;
  enum class filter;
  enum class brush_type;
  enum class scaling;
  enum class refresh_rate;

  class rectangle;
  constexpr bool operator==(const rectangle& lhs, const rectangle& rhs)
    noexcept;
  constexpr bool operator!=(const rectangle& lhs, const rectangle& rhs)
    noexcept;
  class circle;
  constexpr bool operator==(const circle& lhs, const circle& rhs) noexcept;
  constexpr bool operator!=(const circle& lhs, const circle& rhs) noexcept;

  class rgba_color;
  constexpr bool operator==(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;
  constexpr bool operator!=(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;

  class vector_2d;
  constexpr bool operator==(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr bool operator!=(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr vector_2d operator+(const vector_2d& lhs) noexcept;
  constexpr vector_2d operator+(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr vector_2d operator-(const vector_2d& lhs) noexcept;
  constexpr vector_2d operator-(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr vector_2d operator*(const vector_2d& lhs, float rhs) noexcept;
  constexpr vector_2d operator*(float lhs, const vector_2d& rhs) noexcept;

  class matrix_2d;
```

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

namespace path_data {
  class abs_new_path;
  constexpr bool operator==(const abs_new_path&, const abs_new_path&)
    noexcept;
  constexpr bool operator!=(const abs_new_path&, const abs_new_path&)
    noexcept;
  class rel_new_path;
  constexpr bool operator==(const rel_new_path&, const rel_new_path&)
    noexcept;
  constexpr bool operator!=(const rel_new_path&, const rel_new_path&)
    noexcept;
  class close_path;
  constexpr bool operator==(const close_path&, const close_path&) noexcept;
  constexpr bool operator!=(const close_path&, const close_path&) noexcept;
  class set_matrix;
  constexpr bool operator==(const set_matrix&, const set_matrix&) noexcept;
  constexpr bool operator!=(const set_matrix&, const set_matrix&) noexcept;
  class modify_matrix;
  constexpr bool operator==(const modify_matrix&, const modify_matrix&)
    noexcept;
  constexpr bool operator!=(const modify_matrix&, const modify_matrix&)
    noexcept;
  class revert_matrix;
  constexpr bool operator==(const revert_matrix&, const revert_matrix&)
    noexcept;
  constexpr bool operator!=(const revert_matrix&, const revert_matrix&)
    noexcept;
  class abs_cubic_curve;
  constexpr bool operator==(const abs_cubic_curve&, const abs_cubic_curve&)
    noexcept;
  constexpr bool operator!=(const abs_cubic_curve&, const abs_cubic_curve&)
    noexcept;
  class abs_line;
  constexpr bool operator==(const abs_line&, const abs_line&) noexcept;
  constexpr bool operator!=(const abs_line&, const abs_line&) noexcept;
  class abs_move;
  constexpr bool operator==(const abs_move&, const abs_move&) noexcept;
  constexpr bool operator!=(const abs_move&, const abs_move&) noexcept;
  class abs_quadratic_curve;
  constexpr bool operator==(const abs_quadratic_curve&,
    const abs_quadratic_curve&) noexcept;
  constexpr bool operator!=(const abs_quadratic_curve&,
    const abs_quadratic_curve&) noexcept;
  class arc;
  constexpr bool operator==(const arc&, const arc&) noexcept;
  constexpr bool operator!=(const arc&, const arc&) noexcept;
  class rel_cubic_curve;
  constexpr bool operator==(const rel_cubic_curve&, const rel_cubic_curve&)
```

```
      noexcept;
    constexpr bool operator!=(const rel_cubic_curve&, const rel_cubic_curve&)
      noexcept;
    class rel_line;
    constexpr bool operator==(const rel_line&, const rel_line&) noexcept;
    constexpr bool operator!=(const rel_line&, const rel_line&) noexcept;
    class rel_move;
    constexpr bool operator==(const rel_move&, const rel_move&) noexcept;
    constexpr bool operator!=(const rel_move&, const rel_move&) noexcept;
    class rel_quadratic_curve;
    constexpr bool operator==(const rel_quadratic_curve&,
      const rel_quadratic_curve&) noexcept;
    constexpr bool operator!=(const rel_quadratic_curve&,
      const rel_quadratic_curve&) noexcept;
    using path_item = variant<abs_cubic_curve, abs_line, abs_matrix,
      abs_new_path, abs_quadratic_curve, arc, close_path,
      rel_cubic_curve, rel_line, rel_matrix, rel_new_path, rel_quadratic_curve,
      revert_matrix>;
  };

  class path_group;

  template <class Allocator = allocator<path_data::path_data_types>>
  class path_builder;

  template <class Allocator>
  bool operator==(const path_builder<Allocator>& lhs,
    const path_builder<Allocator>& rhs) noexcept;
  template <class Allocator>
  bool operator!=(const path_builder<Allocator>& lhs,
    const path_builder<Allocator>& rhs) noexcept;

  template <class Allocator>
  void swap(path_builder<Allocator>& lhs, path_builder<Allocator>& rhs)
    noexcept(noexcept(lhs.swap(rhs)));

  class brush;

  class render_props;
  class brush_props;
  class clip_props;
  class stroke_props;
  class mask_props;

  class surface;
  class image_surface;
  class display_surface;
  class mapped_surface;

  template <class T>
  constexpr T pi = T(3.141592653589793238462643383279502884L);

  template <class T>
  constexpr T two_pi = T(6.283185307179586476925286766559005768L);
```

```
  template <class T>
  constexpr T half_pi = T(1.57079632679489661923132169163975144L);

  template <class T>
  constexpr T three_pi_over_two = T(4.71238898038468985769396507491925432L);

  int format_stride_for_width(format format, int width) noexcept;
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat,
    scaling scl = scaling::letterbox);
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat, error_code& ec,
    scaling scl = scaling::letterbox) noexcept;
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, scaling scl = scaling::letterbox);
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, ::std::error_code& ec,
    scaling scl = scaling::letterbox) noexcept;
  image_surface make_image_surface(format format, int width, int height);
  image_surface make_image_surface(format format, int width, int height,
    error_code& ec) noexcept;
  image_surface make_image_surface(image_surface& sfc) noexcept;
  float angle_for_point(const vector_2d& ctr, const vector_2d& pt,
    const vector_2d& scl = vector_2d{ 1.0f, 1.0f }) noexcept;
  vector_2d point_for_angle(float ang, float mgn = 1.0f) noexcept;
  vector_2d point_for_angle(float ang, const vector_2d& rad) noexcept;
  vector_2d arc_start(const vector_2d& ctr, float sang, const vector_2d& rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
  vector_2d arc_center(const vector_2d& cpt, float sang, const vector_2d& rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
  vector_2d arc_end(const vector_2d& cpt, float eang, const vector_2d& rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
} } } }
```

6 Colors [io2d.colors]

6.1 Introduction to color [io2d.colors.intro]

1 Color involves many disciplines and has been the subject of many papers, treatises, experiments, studies, and research work in general.

2 While color is an important part of computer graphics, it is only necessary to understand a few concepts from the study of color for computer graphics.

3 A color model defines color mathematically without regard to how humans actually perceive color. These color models are composed of some combination of channels which each channel representing alpha or an ideal color. Color models are useful for working with color computationally, such as in composing operations, because their channel values are homogeneously spaced.

4 A color space, for purposes of computer graphics, is the result of mapping the ideal color channels from a color model, after making any necessary adjustment for alpha, to color channels that are calibrated to align with human perception of colors. Since the perception of color varies from person to person, color spaces use the science of colorimetry to define those perceived colors in order to obtain uniformity to the extent possible. As such, the uniform display of the colors in a color space on different output devices is possible. The values of color channels in a color space are not necessarily homogeneously spaced because of human perception of color.

5 Color models are often termed *linear* while color spaces are often termed *gamma corrected*. The mapping of a color model, such as the RGB color model, to a color space, such as the sRGB color space, is often the application of gamma correction.

6 Gamma correction is the process of transforming homogeneously spaced visual data to visual data that, when displayed, matches the intent of the untransformed visual data.

7 For example a color that is 50% of the maximum intensity of red when encoded as homogeneously spaced visual data, will likely have a different intensity value when it has been gamma corrected so that a human looking at on a computer display will see it as being 50% of the maximum intensity of red that the computer display is capable of producing. Without gamma correction, it would likely have appeared as though it was closer to the maximum intensity than the untransformed data intended it to be.

8 In addition to color channels, colors in computer graphics often have an alpha channel. The value of the alpha channel represents transparency of the color channels when they are combined with other visual data using certain composing algorithms. When using alpha, it should be used in a premultiplied format in order to obtain the desired results when applying multiple composing algorithms that utilize alpha.

6.2 Color usage requirements [io2d.colors.reqs]

1 The use of color throughout this Technical Specification assumes that during rendering and composing operations, color data is linear and that it is in premultiplied format if it has both color and alpha channels.

6.3 Class `rgba_color` [io2d.rgbacolor]

1 The class `rgba_color` describes a four channel color in premultiplied format.

2 There are three color channels, red, green, and blue, each of which is a `float`.

3 There is also an alpha channel, which is a `float`.

4 Legal values for each channel are in the range `[0.0f, 1.0f]`.

6.3.1 rgba_color synopsis [io2d.rgbacolor.synopsis]

```
namespace std::experimental::io2d::v1 {
  class rgba_color {
    // 6.3.2, construct/copy/move/destroy:
    constexpr rgba_color() noexcept;
    template <class T>
    constexpr rgba_color(T r, T g, T b, T a = static_cast<T>(0xFF)) noexcept;
    template <class U>
    constexpr rgba_color(U r, U g, U b, U a = static_cast<U>(1.0f)) noexcept;

    // 6.3.3, modifiers:
    constexpr void r(float val) noexcept;
    constexpr void g(float val) noexcept;
    constexpr void b(float val) noexcept;
    constexpr void a(float val) noexcept;

    // 6.3.4, observers:
    constexpr float r() const noexcept;
    constexpr float g() const noexcept;
    constexpr float b() const noexcept;
    constexpr float a() const noexcept;

    // 6.3.5, static members:
    static const rgba_color alice_blue;
    static const rgba_color antique_white;
    static const rgba_color aqua;
    static const rgba_color aquamarine;
    static const rgba_color azure;
    static const rgba_color beige;
    static const rgba_color bisque;
    static const rgba_color black;
    static const rgba_color blanched_almond;
    static const rgba_color blue;
    static const rgba_color blue_violet;
    static const rgba_color brown;
    static const rgba_color burly_wood;
    static const rgba_color cadet_blue;
    static const rgba_color chartreuse;
    static const rgba_color chocolate;
    static const rgba_color coral;
    static const rgba_color cornflower_blue;
    static const rgba_color cornsilk;
    static const rgba_color crimson;
    static const rgba_color cyan;
    static const rgba_color dark_blue;
    static const rgba_color dark_cyan;
    static const rgba_color dark_goldenrod;
    static const rgba_color dark_gray;
    static const rgba_color dark_green;
    static const rgba_color dark_grey;
    static const rgba_color dark_khaki;
    static const rgba_color dark_magenta;
    static const rgba_color dark_olive_green;
    static const rgba_color dark_orange;
    static const rgba_color dark_orchid;
```

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

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

```
    static const rgba_color violet;
    static const rgba_color wheat;
    static const rgba_color white;
    static const rgba_color white_smoke;
    static const rgba_color yellow;
    static const rgba_color yellow_green;
  };

  // 6.3.6, non-member operators:
  constexpr bool operator==(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;
  constexpr bool operator!=(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;
}
```

6.3.2 rgba_color constructors and assignment operators [io2d.rgbacolor.cons]

```
constexpr rgba_color() noexcept;
```

1 *Effects:* Equivalent to: `rgba_color {0.0f, 0.0f, 0.0f. 0.0f};`.

```
template <class T>
constexpr rgba_color(T r, T g, T b, T a = static_cast<T>(255)) noexcept;
```

2 *Requires:* `is_integral_v<T> == true` and `r >= 0` and `r <= 255` and `g >= 0` and `g <= 255` and `b >= 0` and `b <= 255` and `a >= 0` and `a <= 255`.

3 *Effects:* Constructs an object of type `rgba_color`.

4 *Remarks:* This constructor shall not participate in overload resolution unless `is_integral_v<T>` is `true`.

1. The alpha channel shall be set to the value of `a`.
2. The red channel shall be set to `r` multiplied by the value of `a`.
3. The green channel shall be set to `g` multiplied by the value of `a`.
4. The blue channel shall be set to `b` multiplied by the value of `a`.

```
template <class U>
constexpr rgba_color(U r, U g, U b, U a = static_cast<U>(1.0f)) noexcept;
```

5 *Requires:* `r >= 0.0f` and `r <= 1.0f` and `g >= 0.0f` and `g <= 1.0f` and `b >= 0.0f` and `b <= 1.0f` and `a >= 0.0f` and `a <= 1.0f`.

6 *Effects:* Constructs an object of type `rgba_color`.

1. The alpha channel shall be set to the value of `a / 255.0f`.
2. The red channel shall be set to `r / 255.0f` multiplied by the value of `a / 255.0f`.
3. The green channel shall be set to `g / 255.0f` multiplied by the value of `a / 255.0f`.
4. The blue channel shall be set to `b / 255.0f` multiplied by the value of `a / 255.0f`.

6.3.3 rgba_color modifiers [io2d.rgbacolor.modifiers]

```
constexpr void r(float val) noexcept;
```

1 *Requires:* `val >= 0.0f` and `val <= 1.0f`.

2 *Effects:* The red channel shall be set to `val` multiplied by the value of `a()`.

```
constexpr void g(float val) noexcept;
```

3 *Requires:* `val >= 0.0f` and `val <= 1.0f`.

4 *Effects:* The green channel shall be set to `val` multiplied by the value of `a()`.

```
constexpr void b(float val) noexcept;
```

5 *Requires:* `val >= 0.0f` and `val <= 1.0f`.

6 *Effects:* The blue channel shall be set to `val` multiplied by the value of `a()`.

```
constexpr void a(float val) noexcept;
```

7 *Requires:* `val >= 0.0f` and `val <= 1.0f`.

8 *Effects:*

1. `b((b() / a()) * val);`
2. `g((g() / a()) * val);`
3. `r((r() / a()) * val);`
4. The alpha channel shall be set to `val`.

6.3.4 `rgba_color` observers [io2d.rgbacolor.observers]

```
constexpr float r() const noexcept;
```

1 *Returns:* The value of the red channel.

```
constexpr float g() const noexcept;
```

2 *Returns:* The value of the green channel.

```
constexpr float b() const noexcept;
```

3 *Returns:* The value of the blue channel.

```
constexpr float a() const noexcept;
```

4 *Returns:* The value of the alpha channel.

6.3.5 `rgba_color` static members [io2d.rgbacolor.statics]

1 The `alpha` value of all of the predefined `rgba_color` static members in Table 1 is `255` except for `transparent_-black`, which has an `alpha` value of `0`.

Table 1 — `rgba_color` static members values

Member name	red	green	blue
`alice_blue`	240	248	255
`antique_white`	250	235	215
`aqua`	0	255	255
`aquamarine`	127	255	212
`azure`	240	255	255
`beige`	245	245	220
`bisque`	255	228	196
`black`	0	0	0
`blanched_almond`	255	235	205
`blue`	0	0	255
`blue_violet`	138	43	226
`brown`	165	42	42

Table 1 — `rgba_color` static members values (continued)

Member name	red	green	blue
burly_wood	222	184	135
cadet_blue	95	158	160
chartreuse	127	255	0
chocolate	210	105	30
coral	255	127	80
cornflower_blue	100	149	237
cornsilk	255	248	220
crimson	220	20	60
cyan	0	255	255
dark_blue	0	0	139
dark_cyan	0	139	139
dark_goldenrod	184	134	11
dark_gray	169	169	169
dark_green	0	100	0
dark_grey	169	169	169
dark_khaki	189	183	107
dark_magenta	139	0	139
dark_olive_green	85	107	47
dark_orange	255	140	0
dark_orchid	153	50	204
dark_red	139	0	0
dark_salmon	233	150	122
dark_sea_green	143	188	142
dark_slate_blue	72	61	139
dark_slate_gray	47	79	79
dark_slate_grey	47	79	79
dark_turquoise	0	206	209
dark_violet	148	0	211
deep_pink	255	20	147
deep_sky_blue	0	191	255
dim_gray	105	105	105
dim_grey	105	105	105
dodger_blue	30	144	255
firebrick	178	34	34
floral_white	255	250	240
forest_green	34	139	34
fuchsia	255	0	255
gainsboro	220	220	220
ghost_white	248	248	248
gold	255	215	0
goldenrod	218	165	32
gray	128	128	128
green	0	128	0
green_yellow	173	255	47
grey	128	128	128
honeydew	240	255	240
hot_pink	255	105	180

Table 1 — `rgba_color` static members values (continued)

Member name	red	green	blue
indian_red	205	92	92
indigo	75	0	130
ivory	255	255	240
khaki	240	230	140
lavender	230	230	250
lavender_blush	255	240	245
lawn_green	124	252	0
lemon_chiffon	255	250	205
light_blue	173	216	230
light_coral	240	128	128
light_cyan	224	255	255
light_goldenrod_yellow	250	250	210
light_gray	211	211	211
light_green	144	238	144
light_grey	211	211	211
light_pink	255	182	193
light_salmon	255	160	122
light_sea_green	32	178	170
light_sky_blue	135	206	250
light_slate_gray	119	136	153
light_slate_grey	119	136	153
light_steel_blue	176	196	222
light_yellow	255	255	224
lime	0	255	0
lime_green	50	205	50
linen	250	240	230
magenta	255	0	255
maroon	128	0	0
medium_aquamarine	102	205	170
medium_blue	0	0	205
medium_orchid	186	85	211
medium_purple	147	112	219
medium_sea_green	60	179	113
medium_slate_blue	123	104	238
medium_spring_green	0	250	154
medium_turquoise	72	209	204
medium_violet_red	199	21	133
midnight_blue	25	25	112
mint_cream	245	255	250
misty_rose	255	228	225
moccasin	255	228	181
navajo_white	255	222	173
navy	0	0	128
old_lace	253	245	230
olive	128	128	0
olive_drab	107	142	35
orange	255	69	0

Table 1 — `rgba_color` static members values (continued)

Member name	red	green	blue
orange_red	255	69	0
orchid	218	112	214
pale_goldenrod	238	232	170
pale_green	152	251	152
pale_turquoise	175	238	238
pale_violet_red	219	112	147
papaya_whip	255	239	213
peach_puff	255	218	185
peru	205	133	63
pink	255	192	203
plum	221	160	221
powder_blue	176	224	230
purple	128	0	128
red	255	0	0
rosy_brown	188	143	143
royal_blue	65	105	225
saddle_brown	139	69	19
salmon	250	128	114
sandy_brown	244	164	96
sea_green	46	139	87
sea_shell	255	245	238
sienna	160	82	45
silver	192	192	192
sky_blue	135	206	235
slate_blue	106	90	205
slate_gray	112	128	144
slate_grey	112	128	144
snow	255	250	250
spring_green	0	255	127
steel_blue	70	130	180
tan	210	180	140
teal	0	128	128
thistle	216	191	216
tomato	255	99	71
transparent_black	0	0	0
turquoise	64	244	208
violet	238	130	238
wheat	245	222	179
white	255	255	255
white_smoke	245	245	245
yellow	255	255	0
yellow_green	154	205	50

6.3.6 `rgba_color` non-member operators [io2d.rgbacolor.ops]

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

1 *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) noexcept;
```

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

7 Linear algebra [io2d.linearalgebra]

7.1 Class vector_2d [io2d.vector2d]

7.1.1 vector_2d description [io2d.vector2d.intro]

1 The class `vector_2d` is used as both a point and as a two-dimensional Euclidian vector.

2 It has an *x coordinate* of type `float` and a *y coordinate* of type `float`.

7.1.2 vector_2d synopsis [io2d.vector2d.synopsis]

```
namespace std::experimental::io2d::v1 {
  class vector_2d {
  public:
    // 7.1.3, constructors:
    constexpr vector_2d() noexcept;
    constexpr vector_2d(float x, float y) noexcept;

    // 7.1.4, modifiers:
    constexpr void x(float val) noexcept;
    constexpr void y(float val) noexcept;

    // 7.1.5, observers:
    constexpr float x() const noexcept;
    constexpr float y() const noexcept;
    constexpr float dot(const vector_2d& other) const noexcept;
    float magnitude() const noexcept;
    constexpr float magnitude_squared() const noexcept;
    float angular_direction(const vector_2d& to) const noexcept
    vector_2d to_unit() const noexcept;

    // 7.1.6, member operators:
    constexpr vector_2d& operator+=(const vector_2d& rhs) noexcept;
    constexpr vector_2d& operator-=(const vector_2d& rhs) noexcept;
    constexpr vector_2d& operator*=(float rhs) noexcept;
    constexpr vector_2d& operator*=(const vector_2d& rhs) noexcept;
    constexpr vector_2d& operator/=(float rhs) noexcept;
    constexpr vector_2d& operator/=(const vector_2d& rhs) noexcept;
  };

  // 7.1.7, non-member operators:
  constexpr bool operator==(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr bool operator!=(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr vector_2d operator+(const vector_2d& lhs) noexcept;
  constexpr vector_2d operator+(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr vector_2d operator-(const vector_2d& lhs) noexcept;
  constexpr vector_2d operator-(const vector_2d& lhs, const vector_2d& rhs)
    noexcept;
  constexpr vector_2d operator*(const vector_2d& lhs, float rhs) noexcept;
```

```
    constexpr vector_2d operator*(float lhs, const vector_2d& rhs) noexcept;
    constexpr vector_2d operator*(const vector_2d& lhs, const vector_2d& rhs)
      noexcept;
    constexpr vector_2d operator/(const vector_2d& lhs, float rhs) noexcept;
    constexpr vector_2d operator/(float lhs, const vector_2d& rhs) noexcept;
    constexpr vector_2d operator/(const vector_2d& lhs, const vector_2d& rhs)
      noexcept;
  }
```

7.1.3 `vector_2d` constructors [io2d.vector2d.cons]

`constexpr vector_2d() noexcept;`

1 *Effects:* Equivalent to `vector_2d{ 0.0f, 0.0f }`.

`constexpr vector_2d(float x, float y) noexcept;`

2 *Effects:* Constructs an object of type `vector_2d`.

3 The x coordinate is `x`.

4 The y coordinate is `y`.

7.1.4 `vector_2d` modifiers [io2d.vector2d.modifiers]

`constexpr void x(float val) noexcept;`

1 *Effects:* The x coordinate is `val`.

`constexpr void y(float val) noexcept;`

2 *Effects:* The y coordinate is `val`.

7.1.5 `vector_2d` observers [io2d.vector2d.observers]

`constexpr float x() const noexcept;`

1 *Returns:* The x coordinate.

`constexpr float y() const noexcept;`

2 *Returns:* The y coordinate.

`constexpr float dot(const vector_2d& other) const noexcept;`

3 *Returns:* `x() * other.x() + y() * other.y()`.

`float magnitude() const noexcept;`

4 *Returns:* Equivalent to: `sqrt(dot(*this));`

`constexpr float magnitude_squared() const noexcept;`

5 *Returns:* Equivalent to: `dot(*this);`

`float angular_direction() const noexcept`

6 *Returns:* `atan2(y(), x())` if it is greater than or equal to `0.0f`.

7 Otherwise, `atan2(y(), x()) + two_pi<float>`.

8 [*Note:* The purpose of adding `two_pi<float>` if the result is negative is to produce values in the range `[0.0f, two_pi<float>)`. — *end note*]

`vector_2d to_unit() const noexcept;`

9 *Returns:* `vector_2d{ x() / magnitude(), y() / magnitude()}`.

7.1.6 vector_2d member operators [io2d.vector2d.member.ops]

```
constexpr vector_2d& operator+=(const vector_2d& rhs) noexcept;
```

1 *Effects:* `*this = *this + rhs`.

2 *Returns:* `*this`.

```
constexpr vector_2d& operator-=(const vector_2d& rhs) noexcept;
```

3 *Effects:* Equivalent to: `*this = *this - rhs;`.

4 *Returns:* `*this`.

```
constexpr vector_2d& operator*=(float rhs) noexcept;
constexpr vector_2d& operator*=(const vector_2d& rhs) noexcept;
```

5 *Effects:* Equivalent to: `*this = *this * rhs;`.

6 *Returns:* `*this`.

```
constexpr vector_2d& operator/=(float rhs) noexcept;
constexpr vector_2d& operator/=(const vector_2d& rhs) noexcept;
```

7 *Effects:* Equivalent to: `*this = *this / rhs;`.

8 *Returns:* `*this`.

7.1.7 vector_2d non-member operators [io2d.vector2d.ops]

```
constexpr bool operator==(const vector_2d& lhs, const vector_2d& rhs) noexcept;
```

1 *Returns:* `lhs.x() == rhs.x() && lhs.y() == rhs.y()`.

```
constexpr bool operator!=(const vector_2d& lhs, const vector_2d& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

```
constexpr vector_2d operator+(const vector_2d& lhs) noexcept;
```

3 *Returns:* `lhs`.

```
constexpr vector_2d operator+(const vector_2d& lhs, const vector_2d& rhs)
  noexcept;
```

4 *Returns:* `vector_2d{ lhs.x() + rhs.x(), lhs.y() + rhs.y() }`.

```
constexpr vector_2d operator-(const vector_2d& lhs) noexcept;
```

5 *Returns:* `vector_2d{ -lhs.x(), -lhs.y() }`.

```
constexpr vector_2d operator-(const vector_2d& lhs, const vector_2d& rhs)
  noexcept;
```

6 *Returns:* `vector_2d{ lhs.x() - rhs.x(), lhs.y() - rhs.y() }`.

```
constexpr vector_2d operator*(const vector_2d& lhs, const vector_2d& rhs)
  noexcept;
```

7 *Returns:* `vector_2d{ lhs.x() * rhs.x(), lhs.y() * rhs.y() }`.

```
constexpr vector_2d operator*(const vector_2d& lhs, float rhs) noexcept;
```

8 *Returns:* `vector_2d{ lhs.x() * rhs, lhs.y() * rhs }`.

```
constexpr vector_2d operator*(float lhs, const vector_2d& rhs) noexcept;
```

9 *Returns:* `vector_2d{ lhs * rhs.x(), lhs * rhs.y() }`.

```
constexpr vector_2d operator/(const vector_2d& lhs, const vector_2d& rhs)
  noexcept;
```

10 *Requires:* `rhs.x()` is not `0.0f` and `rhs.y()` is not `0.0f`.

11 *Returns:* `vector_2d{ lhs.x() / rhs.x(), lhs.y() / rhs.y() }`.

```
constexpr vector_2d operator/(const vector_2d& lhs, float rhs) noexcept;
```

12 *Requires:* `rhs` is not `0.0f`.

13 *Returns:* `vector_2d{ lhs.x() / rhs, lhs.y() / rhs }`.

```
constexpr vector_2d operator/(float lhs, const vector_2d& rhs) noexcept;
```

14 *Requires:* `rhs.x()` is not `0.0f` and `rhs.y()` is not `0.0f`.

15 *Returns:* `vector_2d{ lhs / rhs.x(), lhs / rhs.y() }`.

7.2 Class `matrix_2d` [io2d.matrix2d]

7.2.1 `matrix_2d` description [io2d.matrix2d.intro]

1 The `matrix_2d` class represents a three row by three column matrix. Its purpose is to perform affine transformations.

2 The layout of the matrix is:
[[$M00$ $M01$ $M02$]]
[[$M10$ $M11$ $M12$]]
[[$M20$ $M21$ $M22$]]

3 The value of M02 is `0.0f`. The value of M12 is `0.0f`. The value of M22 is `1.0f`. None of the operations on a `matrix_2d` object can modify the values of M02, M12, and M22. As such they are not part of the observable state of a `matrix_2d` object.

4 The performance of any mathematical operation upon a `matrix_2d` shall be carried out as-if the omitted third column data members were present with the values prescribed in the previous paragraph.

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

```
namespace std::experimental::io2d::v1 {
  class matrix_2d {
  public:
    // 7.2.3, construct:
    constexpr matrix_2d() noexcept;
    constexpr matrix_2d(float v00, float v01, float v10, float v11,
      float v20, float v21) noexcept;

    // 7.2.4, static factory functions:
    constexpr static matrix_2d init_translate(const vector_2d& value) noexcept;
    constexpr static matrix_2d init_scale(const vector_2d& value) noexcept;
    static matrix_2d init_rotate(float radians) noexcept;
    static matrix_2d init_reflect(float radians) noexcept;
    constexpr static matrix_2d init_shear_x(float factor) noexcept;
    constexpr static matrix_2d init_shear_y(float factor) noexcept;

    // 7.2.5, modifiers:
    constexpr void m00(float v) noexcept;
```

```
    constexpr void m01(float v) noexcept;
    constexpr void m10(float v) noexcept;
    constexpr void m11(float v) noexcept;
    constexpr void m20(float v) noexcept;
    constexpr void m21(float v) noexcept;
    constexpr matrix_2d& translate(const vector_2d& v) noexcept;
    constexpr matrix_2d& scale(const vector_2d& v) noexcept;
    matrix_2d& rotate(float radians) noexcept;
    matrix_2d& reflect(float radians) noexcept;
    constexpr matrix_2d& shear_x(float factor) noexcept;
    constexpr matrix_2d& shear_y(float factor) noexcept;

    // 7.2.6, observers:
    constexpr float m00() const noexcept;
    constexpr float m01() const noexcept;
    constexpr float m10() const noexcept;
    constexpr float m11() const noexcept;
    constexpr float m20() const noexcept;
    constexpr float m21() const noexcept;
    constexpr bool is_finite() const noexcept;
    constexpr bool is_invertible() const noexcept;
    constexpr float determinant() const noexcept;
    constexpr matrix_2d inverse() const noexcept;
    constexpr vector_2d transform_point(const vector_2d& pt) const noexcept;

    // 7.2.7, matrix_2d member operators:
    constexpr matrix_2d& operator*=(const matrix_2d& rhs) noexcept;
  };

  // 7.2.8, matrix_2d non-member operators:
  constexpr matrix_2d operator*(const matrix_2d& lhs, const matrix_2d& rhs)
    noexcept;
  constexpr bool operator==(const matrix_2d& lhs, const matrix_2d& rhs)
    noexcept;
  constexpr bool operator!=(const matrix_2d& lhs, const matrix_2d& rhs)
    noexcept;
  constexpr vector_2d operator*(const vector_2d& v, const matrix_2d& m)
    noexcept;
}
```

7.2.3 `matrix_2d` constructors [io2d.matrix2d.cons]

`constexpr matrix_2d() noexcept;`

1 *Effects:* Equivalent to: `matrix_2d{ 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f }`.

2 [*Note:* The resulting matrix is the identity matrix. — *end note*]

```
constexpr matrix_2d(float v00, float v01, float v10, float v11,
  float v20, float v21) noexcept;
```

3 *Effects:* Constructs an object of type `matrix_2d`.

4 As-if the resulting matrix was:
[[$v00$ $v01$ $0.0f$]]
[[$v10$ $v11$ $0.0f$]]
[[$v20$ $v21$ $1.0f$]]

7.2.4 `matrix_2d` static factory functions [io2d.matrix2d.staticfactories]

`constexpr static matrix_2d init_translate(const vector_2d& value) noexcept;`

1 *Returns:* `matrix(1.0f, 0.0f, 0.0f, 1.0f, value.x(), value.y())`.

`constexpr static matrix_2d init_scale(const vector_2d& value) noexcept;`

2 *Returns:* `matrix(value.x(), 0.0f, 0.0f, value.y(), 0.0f, 0.0f)`.

`static matrix_2d init_rotate(float radians) noexcept;`

3 *Returns:* `matrix(cos(radians), -sin(radians), sin(radians), cos(radians), 0.0f, 0.0f)`.

`static matrix_2d init_reflect(float radians) noexcept;`

4 *Returns:*

matrix(cos(radians * 2.0f), sin(radians * 2.0f), sin(radians * 2.0f), -cos(radians * 2.0f), 0.0f, 0.0f)

`constexpr static matrix_2d init_shear_x(float factor) noexcept;`

5 *Returns:* `matrix(1.0f, 0.0f, factor, 1.0f, 0.0f, 0.0f)`.

`constexpr static matrix_2d init_shear_y(float factor) noexcept;`

6 *Returns:* `matrix{ 1.0f, factor, 0.0f, 1.0f, 0.0f, 0.0f }`

7.2.5 `matrix_2d` modifiers [io2d.matrix2d.modifiers]

`constexpr void m00(float val) noexcept;`

1 *Effects:* M00 is `val`.

`constexpr void m01(float val) noexcept;`

2 *Effects:* M01 is `val`.

`constexpr void m10(float val) noexcept;`

3 *Effects:* M10 is `val`.

`constexpr void m11(float val) noexcept;`

4 *Effects:* M11 is `val`.

`constexpr void m20(float val) noexcept;`

5 *Effects:* M20 is `val`.

`constexpr void m21(float val) noexcept;`

6 *Effects:* M21 is `val`.

`constexpr matrix_2d& translate(const vector_2d& val) noexcept;`

7 *Effects:* Equivalent to: `*this = *this * init_translate(val);`

8 *Returns:* `*this`.

`constexpr matrix_2d& scale(const vector_2d& val) noexcept;`

9 *Effects:* Equivalent to: `*this = *this * init_scale(val);`

10 *Returns:* `*this`.

`matrix_2d& rotate(float radians) noexcept;`

11 *Effects:* Equivalent to: `*this = *this * init_rotate(radians);`

12 *Returns:* `*this`.

`matrix_2d& reflect(float radians) noexcept;`

13 *Effects:* Equivalent to: `*this = *this * init_reflect(radians);`

14 *Returns:* `*this`.

`constexpr matrix_2d& shear_x(float factor) noexcept;`

15 *Effects:* Equivalent to: `*this = *this * init_shear_x(factor);`

16 *Returns:* `*this`.

`constexpr matrix_2d& shear_y(float factor) noexcept;`

17 *Effects:* Equivalent to: `*this = *this * init_shear_y(factor);`

18 *Returns:* `*this`.

7.2.6 `matrix_2d` observers [io2d.matrix2d.observers]

`constexpr float m00() const noexcept;`

1 *Returns:* M00.

`constexpr float m01() const noexcept;`

2 *Returns:* M01.

`constexpr float m10() const noexcept;`

3 *Returns:* M10.

`constexpr float m11() const noexcept;`

4 *Returns:* M11.

`constexpr float m20() const noexcept;`

5 *Returns:* M20.

`constexpr float m21() const noexcept;`

6 *Returns:* M21.

`constexpr bool is_finite const noexcept;`

7 *Returns:* `true` if the observable behavior of all of the following expressions evaluates to true:

(7.1) — `isfinite(m00());`

(7.2) — `isfinite(m01());`

(7.3) — `isfinite(m10());`

(7.4) — `isfinite(m11());`

(7.5) — `isfinite(m20());`

(7.6) — `isfinite(m21());`

8 Otherwise returns `false`.

9 [*Note:* The specification of `isfinite` in N4618 does not include the `constexpr` specifier. Regardless, the requirements stated in [library.c] and [c.math.fpclass] in N4618 make it possible to implement a constexpr function that produces the observable behavior of `isfinite`. As a result, this function can

be implemented as a constexpr function. — *end note*]

`constexpr bool is_invertible() const noexcept;`

10 *Requires:* `is_finite()` is `true`.

11 *Returns:* `determinant() != 0.0f`.

`constexpr matrix_2d inverse() const noexcept;`

12 *Requires:* `is_invertible()` is `true`.

13 *Returns:* Let `inverseDeterminant` be `1.0f / determinant()`.

```
return matrix_2d{
   (_M11 * 1.0f - 0.0f * _M21) * inverseDeterminant,
  -(_M01 * 1.0f - 0.0f * _M21) * inverseDeterminant,
  -(_M10 * 1.0f - 0.0f * _M20) * inverseDeterminant,
   (_M00 * 1.0f - 0.0f * _M20) * inverseDeterminant,
   (_M10 * _M21 - _M11 * _M20) * inverseDeterminant,
  -(_M00 * _M21 - _M01 * _M20) * inverseDeterminant
};
```

`constexpr float determinant() const noexcept;`

14 *Requires:* `is_finite()` is `true`.

15 *Returns:* `m00() * m11() - m01() * m10()`.

`constexpr vector_2d transform_point(const vector_2d& pt) const noexcept;`

16 *Returns:* `vector_2d((m00() * pt.x() + m10() * pt.y()) + m20(), (m01() * pt.x() + m11() * pt.y()) + m21())`.

7.2.7 `matrix_2d` member operators [io2d.matrix2d.member.ops]

`constexpr matrix_2d& operator*=(const matrix_2d& rhs) noexcept;`

1 *Effects:* Equivalent to: *this = *this * rhs;

2 *Returns:* `*this`.

7.2.8 `matrix_2d` non-member operators [io2d.matrix2d.ops]

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

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()
}
```

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

2 *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()
```

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

3 *Returns:* `!(lhs == rhs)`.

```
constexpr vector_2d operator*(const vector_2d& v, const matrix_2d& m) noexcept;
```

4 *Returns:* Equivalent to: `m.transform_point(v)`.

8 Geometry [io2d.geometry]

8.1 Class `rectangle` [io2d.rectangle]

8.1.1 `rectangle` description [io2d.rectangle.intro]

1 The class `rectangle` describes a rectangle.

2 It has an *x coordinate* of type `float`, a *y coordinate* of type `float`, a *width* of type `float`, and a *height* of type `float`.

8.1.2 `rectangle` synopsis [io2d.rectangle.synopsis]

```
namespace std::experimental::io2d::v1 {
  class rectangle {
  public:
    // 8.1.3, construct:
    constexpr rectangle() noexcept;
    constexpr rectangle(float x, float y, float width, float height)
      noexcept;
    constexpr rectangle(const vector_2d& tl, const vector_2d& br) noexcept;

    // 8.1.4, modifiers:
    constexpr void x(float val) noexcept;
    constexpr void y(float val) noexcept;
    constexpr void width(float val) noexcept;
    constexpr void height(float val) noexcept;
    constexpr void top_left(const vector_2d& val) noexcept;
    constexpr void bottom_right(const vector_2d& val) noexcept;

    // 8.1.5, observers:
    constexpr float x() const noexcept;
    constexpr float y() const noexcept;
    constexpr float width() const noexcept;
    constexpr float height() const noexcept;
    constexpr vector_2d top_left() const noexcept;
    constexpr vector_2d bottom_right() const noexcept;
  };

  // 8.1.6, operators:
  constexpr bool operator==(const rectangle& lhs, const rectangle& rhs)
    noexcept;
  constexpr bool operator!=(const rectangle& lhs, const rectangle& rhs)
    noexcept;
}
```

8.1.3 `rectangle` constructors [io2d.rectangle.cons]

```
constexpr rectangle() noexcept;
```

1 *Effects:* Equivalent to `rectangle{ 0.0f, 0.0f, 0.0f, 0.0f }`.

```
constexpr rectangle(float x, float y, float w, float h) noexcept;
```

2 *Requires:* `w` is not less than `0.0f` and `h` is not less than `0.0f`.

3 *Effects:* Constructs an object of type `rectangle`.

4 The x coordinate is `x`. The y coordinate is `y`. The width is `w`. The height is `h`.

```
constexpr rectangle(const vector_2d& tl, const vector_2d& br) noexcept;
```

5 *Effects:* Constructs an object of type `rectangle`.

6 The x coordinate is `tl.x()`. The y coordinate is `tl.y()`. The width is `max(0.0f, br.x() - tl.x())`. The height is `max(0.0f, br.y() - tl.y())`.

8.1.4 `rectangle` modifiers [io2d.rectangle.modifiers]

```
constexpr void x(float val) noexcept;
```

1 *Effects:* The x coordinate is `val`.

```
constexpr void y(float val) noexcept;
```

2 *Effects:* The y coordinate is `val`.

```
constexpr void width(float val) noexcept;
```

3 *Effects:* The width is `val`.

```
constexpr void height(float val) noexcept;
```

4 *Effects:* The height is `val`.

```
constexpr void top_left(const vector_2d& val) noexcept;
```

5 *Effects:* The x coordinate is `val.x()`.

Effects: The y coordinate is `val.y()`.

```
constexpr void bottom_right(const vector_2d& val) noexcept;
```

6 *Effects:* The width is `max(0.0f, val.x() - x())`.

7 The height is `max(0.0f, value.y() - y())`.

8.1.5 `rectangle` observers [io2d.rectangle.observers]

```
constexpr float x() const noexcept;
```

1 *Returns:* The x coordinate.

```
constexpr float y() const noexcept;
```

2 *Returns:* The y coordinate.

```
constexpr float width() const noexcept;
```

3 *Returns:* The width.

```
constexpr float height() const noexcept;
```

4 *Returns:* The height.

```
constexpr vector_2d top_left() const noexcept;
```

5 *Returns:* A `vector_2d` object constructed with the x coordinate as its first argument and the y coordinate as its second argument.

```
constexpr vector_2d bottom_right() const noexcept;
```

6 *Returns:* A `vector_2d` object constructed with the width added to the x coordinate as its first argument and the height added to the y coordinate as its second argument.

8.1.6 `rectangle` operators [io2d.rectangle.ops]

```
constexpr bool operator==(const rectangle& lhs, const rectangle& rhs) noexcept;
```

1 *Returns:*

```
lhs.x() == rhs.x() && lhs.y() == rhs.y() &&
lhs.width() == rhs.width() && lhs.height() == rhs.height()
```

```
constexpr bool operator!=(const rectangle& lhs, const rectangle& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

8.2 Class `circle` [io2d.circle]

8.2.1 `circle` description [io2d.circle.intro]

1 The class `circle` describes a circle.

2 It has a *center* of type `vector_2d` and a *radius* of type `float`.

8.2.2 `circle` synopsis [io2d.circle.synopsis]

```
namespace std::experimental::io2d::v1 {
  class circle {
  public:
    // 8.2.3, constructors:
    constexpr circle() noexcept;
    contexpr circle(const vector_2d& ctr, float rad) noexcept;

    // 8.2.4, modifiers:
    constexpr void center(const vector_2d& ctr) noexcept;
    constexpr void radius(float r) noexcept;

    // 8.2.5, observers:
    constexpr vector_2d center() const noexcept;
    constexpr float radius() const noexcept;
  };
}
```

8.2.3 `circle` constructors [io2d.circle.cons]

```
constexpr circle() noexcept;
```

1 *Effects:* Equivalent to: `circle({ 0.0f, 0.0f }, 0.0f)`.

```
constexpr circle(const vector_2d& ctr, float r) noexcept;
```

Requires: `r >= 0.0f`.

2 *Effects:* Constructs an object of type `circle`.

3 The center is `ctr`. The fadius is `r`.

8.2.4 `circle` modifiers [io2d.circle.modifiers]

```
constexpr void center(const vector_2d& ctr) noexcept;
```

1 *Effects:* The center is `ctr`.

```
constexpr void radius(float r) noexcept;
```

Requires: `r >= 0.0f`.

2 *Effects:* The radius is `r`.

8.2.5 `circle` observers [io2d.circle.observers]

```
constexpr float center() const noexcept;
```

1 *Returns:* The center.

```
constexpr float radius() const noexcept;
```

2 *Returns:* The radius.

9 Paths [io2d.paths]

9.1 Overview of paths [io2d.paths.overview]

1 Paths define geometric objects which can be stroked (Table 18), filled, masked, and used to define a clip area (See: 11.12.1.

2 A path group contains zero or more paths.

3 A path is composed of at least one path segment.

4 A path may contain degenerate path segments. When a path is rendered in certain rendering and composing operations, degenerate path segments can produce observable behavior. [*Example:* When a degenerate path segment is rendered in a stroke rendering and composing operation (see 11.15.7), the `line_cap` value contained in its `stroke_props` argument can result in a degenerate path segment producing observable behavior in the form of a circle or square, or some variation thereof. — *end example*]

5 Paths provide vector graphics functionality. As such they are particularly useful in situations where an application is intended to run on a variety of platforms whose output devices (11.17.1) span a large gamut of sizes, both in terms of measurement units and in terms of a horizontal and vertical pixel count, in that order.

6 A `path_group` object is an immutable resource wrapper containing a path group (9.4). A `path_group` object is created from the paths contained in a `path_builder` object. It can also be default constructed, in which case the `path_group` object contains no paths. [*Note:* `path_group` objects provide significant optimization opportunities for implementations due to being immutable and opaque. — *end note*]

9.2 Path group examples (Informative) [io2d.paths.example]

9.2.1 Overview [io2d.paths.example.intro]

1 Path groups are composed of zero or more paths. The following examples show the basics of how path groups work in practice.

2 Every example is placed within the following code at the indicated spot. This code is shown here once to avoid repetition:

```
#include <experimental/io2d>

using namespace std;
using namespace std::experimental::io2d;

int main() {
  auto imgSfc = make_image_surface(format::argb32, 300, 200);
  brush backBrush{ rgba_color::black };
  brush foreBrush{ rgba_color::white };
  render_props aliased{ antialias::none };
  path_builder<> pb{};
  imgSfc.paint(backBrush);

  // Example code goes here.

  // Example code ends.

  imgSfc.save(filesystem::path("example.png"), image_file_format::png);
  return 0;
```

```
}
```

9.2.2 Example 1 [io2d.paths.examples.one]

1 Example 1 consists of a single path, forming a trapezoid:

```
pb.new_path({ 80.0f, 20.0f }); // Begins the path.
pb.line({ 220.0f, 20.0f }); // Creates a line from the [80, 20] to [220, 20].
pb.rel_line({ 60.0f, 160.0f }); // Line from [220, 20] to
  // [220 + 60, 160 + 20]. The "to" point is relative to the starting point.
pb.rel_line({ -260.0f, 0.0f }); // Line from [280, 180] to
  // [280 - 260, 180 + 0].
pb.close_path(); // Creates a line from [20, 180] to [80, 20]
  // (the new path point), which makes this a closed path.
imgSfc.stroke(foreBrush, pb, nullopt, nullopt, nullopt, aliased);
```

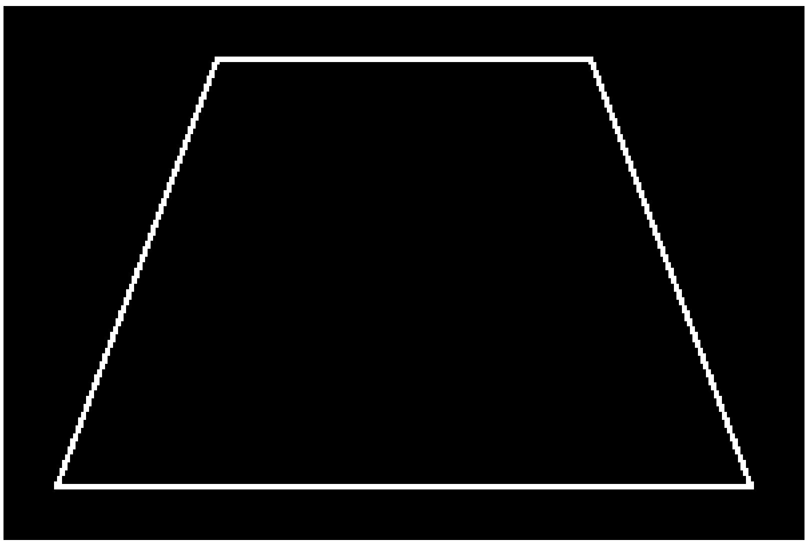

Figure 1 — Example 1 result

9.2.3 Example 2 [io2d.paths.examples.two]

1 Example 2 consists of two paths. The first is a rectangular open path (on the left) and the second is a rectangular closed path (on the right):

```
pb.new_path({ 20.0f, 20.0f }); // Begin the first path.
pb.rel_line({ 100.0f, 0.0f });
pb.rel_line({ 0.0f, 160.0f });
pb.rel_line({ -100.0f, 0.0f });
pb.rel_line({ 0.0f, -160.0f });

pb.new_path({ 180.0f, 20.0f }); // End the first path and begin the second path.
pb.rel_line({ 100.0f, 0.0f });
pb.rel_line({ 0.0f, 160.0f });
pb.rel_line({ -100.0f, 0.0f });
pb.close_path(); // End the second path.
imgSfc.stroke(foreBrush, pb, nullopt, stroke_props{ 10.0f }, nullopt, alised);
```

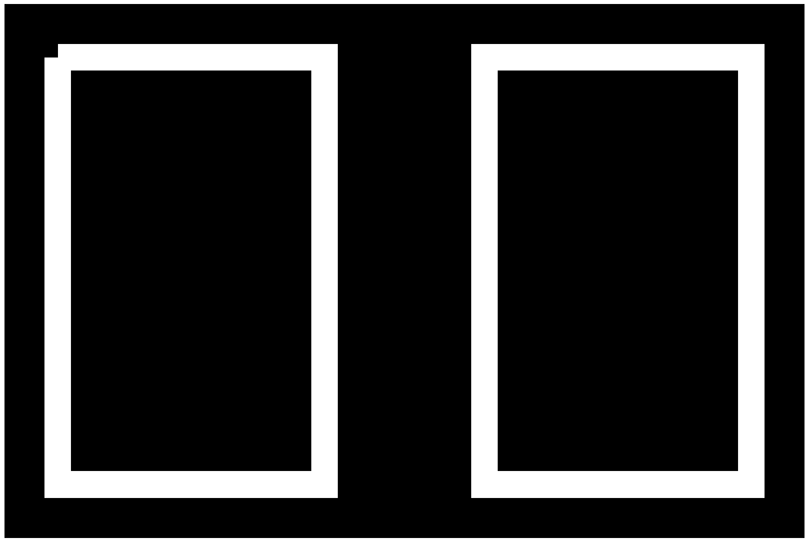

Figure 2 — Example 2 result

2 The resulting image from example 2 shows the difference between an open path and a closed path. Each path begins and ends at the same point. The difference is that with the closed path, that the rendering of the point where the initial path segment and final path segment meet is controlled by the `line_join` value in the `stroke_props` class, which in this case is the default value of `line_join::miter`. In the open path, the rendering of that point receives no special treatment such that each path segment at that point is rendered using the `line_cap` value in the `stroke_props` class, which in this case is the default value of `line_cap::none`.

3 That difference between rendering as a `line_join` versus rendering as two `line_caps` is what causes the notch to appear in the open path segment. Path segments are rendered such that half of the stroke width is rendered on each side of the point being evaluated. With no line cap, each segment begins and ends exactly at the point specified.

4 So for the open path, the first line begins at `vector_2d{ 20.0f, 20.0f }` and the last line ends there. Given the stroke width of `10.0f`, the visible result for the first line is a rectangle with an upper left corner of `vector_2d{ 20.0f, 15.0f }` and a lower right corner of `vector_2d{ 120.0f, 25.0f }`. The last line appears as a rectangle with an upper left corner of `vector_2d{ 15.0f, 20.0f }` and a lower right corner of `vector_2d{ 25.0f, 180.0f }`. This produces the appearance of a square gap between `vector_2d{ 15.0f, 15.0f }` and `vector_2d{20.0f, 20.0f }`.

5 For the closed path, adjusting for the coordinate differences, the rendering facts are the same as for the open path except for one key difference: the point where the first line and last line meet is rendered as a line join rather than two line caps, which, given the default value of `line_join::miter`, produces a miter, adding that square area to the rendering result.

9.2.4 Example 3 [io2d.paths.examples.three]

1 Example 3 demonstrates open and closed paths each containing either a quadratic curve or a cubic curve.

```
pb.new_path({ 20.0f, 20.0f });
pb.rel_quadratic_curve({ 60.0f, 120.0f }, { 60.0f, -120.0f });
pb.rel_new_path({ 20.0f, 0.0f });
pb.rel_quadratic_curve({ 60.0f, 120.0f }, { 60.0f, -120.0f });
pb.close_path();
```

```
pb.new_path({ 20.0f, 150.0f });
pb.rel_cubic_curve({ 40.0f, -120.0f }, { 40.0f, 120.0f * 2.0f },
  { 40.0f, -120.0f });
pb.rel_new_path({ 20.0f, 0.0f });
pb.rel_cubic_curve({ 40.0f, -120.0f }, { 40.0f, 120.0f * 2.0f },
  { 40.0f, -120.0f });
pb.close_path();
imgSfc.stroke(foreBrush, pb, nullopt, nullopt, nullopt, aliased);
```

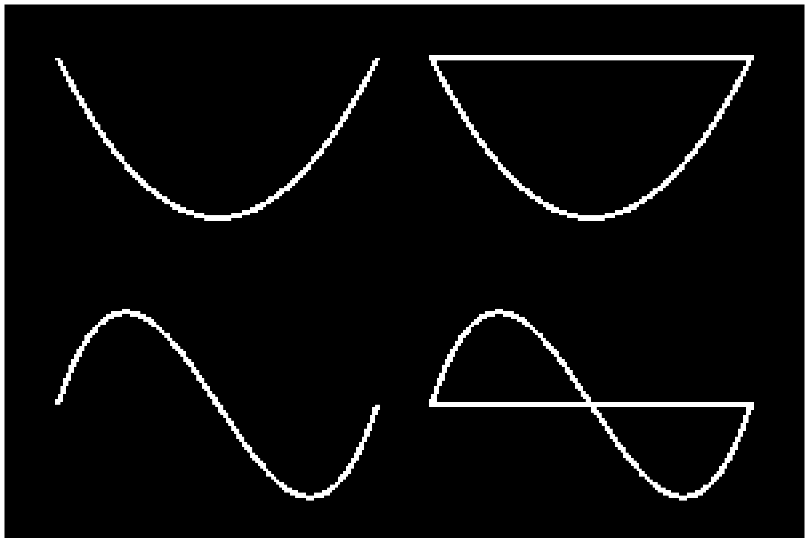

Figure 3 — Path example 3

2 [*Note:* `pb.quadratic_curve({ 80.0f, 140.0f }, { 140.0f, 20.0f });` would be the absolute equivalent of the first curve in example 3. — *end note*]

9.2.5 Example 4 [io2d.paths.examples.four]

1 Example 4 shows how to draw "C++" using paths.

2 For the "C", it is created using an arc. A scaling matrix is used to make it slightly elliptical. It is also desirable that the arc has a fixed center point, `vector_2d{ 85.0f, 100.0f }`. The inverse of the scaling matrix is used in combination with the `point_for_angle` function to determine the point at which the arc should begin in order to get achieve this fixed center point. The "C" is then stroked.

3 Unlike the "C", which is created using an open path that is stroked, each "+" is created using a closed path that is filled. To avoid filling the "C", `pb.clear();` is called to empty the container. The first "+" is created using a series of lines and is then filled.

4 Taking advantage of the fact that `path_builder` is a container, rather than create a brand new path for the second "+", a translation matrix is applied by inserting a `path_data::change_matrix` path item before the `path_data::new_path` object in the existing plus, reverting back to the old matrix immediately after the and then filling it again.

```
// Create the "C".
const matrix_2d scl = matrix_2d::init_scale({ 0.9f, 1.1f });
auto pt = scl.inverse().transform_point({ 85.0f, 100.0f }) +
  point_for_angle(half_pi<float> / 2.0f, 50.0f);
pb.matrix(scl);
```

```
pb.new_path(pt);
pb.arc({ 50.0f, 50.0f }, three_pi_over_two<float>, half_pi<float> / 2.0f);
imgSfc.stroke(foreBrush, pb, nullopt, stroke_props{ 10.0f });
// Create the first "+".
pb.clear();
pb.new_path({ 130.0f, 105.0f });
pb.rel_line({ 0.0f, -10.0f });
pb.rel_line({ 25.0f, 0.0f });
pb.rel_line({ 0.0f, -25.0f });
pb.rel_line({ 10.0f, 0.0f });
pb.rel_line({ 0.0f, 25.0f });
pb.rel_line({ 25.0f, 0.0f });
pb.rel_line({ 0.0f, 10.0f });
pb.rel_line({ -25.0f, 0.0f });
pb.rel_line({ 0.0f, 25.0f });
pb.rel_line({ -10.0f, 0.0f });
pb.rel_line({ 0.0f, -25.0f });
pb.close_path();
imgSfc.fill(foreBrush, pb);
// Create the second "+".
pb.insert(pb.begin(), path_data::change_matrix(
  matrix_2d::init_translate({ 80.0f, 0.0f })));
imgSfc.fill(foreBrush, pb);
```

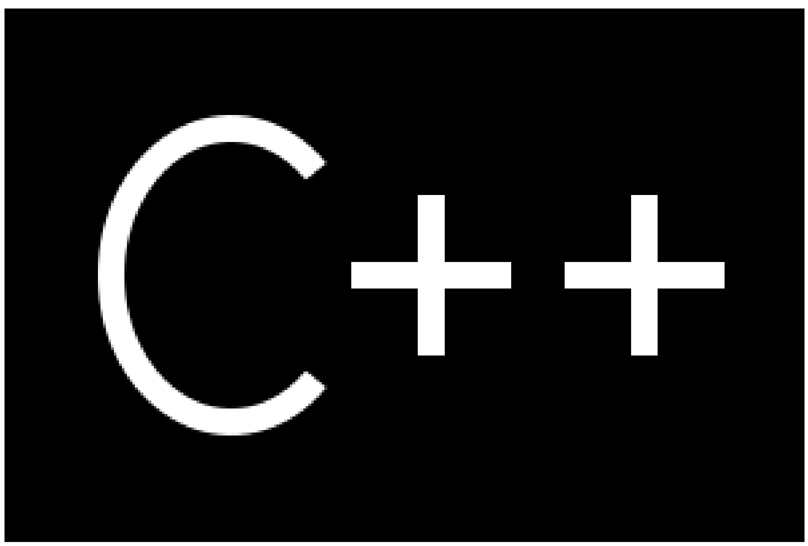

Figure 4 — Path example 4

9.3 Path items [io2d.paths.items]

9.3.1 Introduction [io2d.paths.items.intro]

1 The classes in the `path_data` namespace describe path items.

2 A path begins with an `abs_new_path` or `rel_new_path` object. A path ends when:

(2.1) — a `close_path` object is encountered;

(2.2) — a `abs_new_path` or `rel_new_path` object is encountered; or

(2.3) — there are no more path items in the path's path group.

3 The `path_builder` class is a sequential container that contains a path group. It provides a simple interface for building a path group but a path group can be created using any container that stores `path_data::path_item` objects.

9.3.2 Class `abs_new_path` [io2d.absnewpath]

1 The class `abs_new_path` describes a path item that is a new path instruction.

2 It has an *at point* of type `vector_2d`.

9.3.2.1 `abs_new_path` synopsis [io2d.absnewpath.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class abs_new_path {
    public:
      // 9.3.2.2, construct:
      constexpr abs_new_path() noexcept;
      constexpr explicit abs_new_path(const vector_2d& pt) noexcept;

      // 9.3.2.3, modifiers:
      constexpr void at(const vector_2d& pt) noexcept;

      // 9.3.2.4, observers:
      constexpr vector_2d at() const noexcept;
    };

    // 9.3.2.5, non-members:
    constexpr bool operator==(const abs_new_path& lhs, const abs_new_path& rhs)
      noexcept;
    constexpr bool operator!=(const abs_new_path& lhs, const abs_new_path& rhs)
      noexcept;
  }
}
```

9.3.2.2 `abs_new_path` constructors [io2d.absnewpath.cons]

```
constexpr abs_new_path() noexcept;
```

1 *Effects:* Equivalent to: `abs_new_path{ vector_2d() };`

```
constexpr explicit abs_new_path(const vector_2d& pt) noexcept;
```

2 *Effects:* Constructs an object of type `abs_new_path`.

3 The at point is `pt`.

9.3.2.3 `abs_new_path` modifiers [io2d.absnewpath.modifiers]

```
constexpr void at(const vector_2d& pt) noexcept;
```

1 *Effects:* The at point is `pt`.

9.3.2.4 `abs_new_path` observers [io2d.absnewpath.observers]

```
constexpr vector_2d at() const noexcept;
```

1 *Returns:* The at point.

9.3.2.5 Non-member functions [io2d.absnewpath.nonmember]

```
constexpr bool operator==(const abs_new_path& lhs, const abs_new_path& rhs)
  noexcept;
```

1 *Returns:* `lhs.at() == rhs.at()`.

```
constexpr bool operator!=(const abs_new_path& lhs, const abs_new_path& rhs)
  noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.3 Class `rel_new_path` [io2d.relnewpath]

1 The class `rel_new_path` describes a path item that is a new path instruction.

2 It has an *at point* of type `vector_2d`.

9.3.3.1 `rel_new_path` synopsis [io2d.relnewpath.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class rel_new_path {
    public:
      // 9.3.3.2, construct:
      constexpr rel_new_path() noexcept;
      constexpr explicit rel_new_path(const vector_2d& pt) noexcept;

      // 9.3.3.3, modifiers:
      constexpr void at(const vector_2d& pt) noexcept;

      // 9.3.3.4, observers:
      constexpr vector_2d at() const noexcept;
    };

    // 9.3.3.5, non-members:
    bool operator==(const rel_new_path& lhs, const rel_new_path& rhs) noexcept;
    bool operator!=(const rel_new_path& lhs, const rel_new_path& rhs) noexcept;
  }
}
```

9.3.3.2 `rel_new_path` constructors [io2d.relnewpath.cons]

```
constexpr rel_new_path() noexcept;
```

1 *Effects:* Equivalent to: `rel_new_path{ vector_2d() };`

```
constexpr explicit rel_new_path(const vector_2d& pt) noexcept;
```

2 *Effects:* Constructs an object of type `rel_new_path`.

3 The at point is `pt`.

9.3.3.3 `rel_new_path` modifiers [io2d.relnewpath.modifiers]

```
constexpr void at(const vector_2d& pt) noexcept;
```

1 *Effects:* The at point is `pt`.

9.3.3.4 `rel_new_path` observers [io2d.relnewpath.observers]

```
constexpr vector_2d at() const noexcept;
```

1 *Returns:* The at point.

9.3.3.5 Non-member functions [io2d.relnewpath.nonmember]

```
constexpr bool operator==(const rel_new_path& lhs, const rel_new_path& rhs)
  noexcept;
```

1 *Returns:* `lhs.at() == rhs.at()`.

```
constexpr bool operator!=(const rel_new_path& lhs, const rel_new_path& rhs)
  noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.4 Class close_path [io2d.closepath]

1 The class `close_path` describes a path item that is a close path instruction.

9.3.4.1 close_path synopsis [io2d.closepath.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class close_path {
      constexpr close_path() noexcept;
    };

    // 9.3.4.2, non-members
    constexpr bool operator==(const close_path&, const close_path&) noexcept;
    constexpr bool operator!=(const close_path&, const close_path&) noexcept;
  }
}
```

9.3.4.2 Non-member functions [io2d.closepath.nonmember]

```
constexpr bool operator==(const close_path&, const close_path&) noexcept;
```

1 *Returns:* `true`.

```
constexpr bool operator!=(const close_path&, const close_path&) noexcept;
```

2 *Returns:* `false`.

9.3.5 Class abs_matrix [io2d.absmatrix]

9.3.5.1 abs_matrix synopsis [io2d.absmatrix.synopsis]

1 The class `abs_matrix` describes a path item that is a path group instruction.

2 It has a transform matrix of type `matrix_2d`.

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class abs_matrix {
    public:
      // 9.3.5.2, construct:
      constexpr abs_matrix() noexcept;
      constexpr explicit abs_matrix(const matrix_2d& m) noexcept;

      // 9.3.5.3, modifiers:
      constexpr void matrix(const matrix_2d& m) noexcept;

      // 9.3.5.4, observers:
      constexpr matrix_2d matrix() const noexcept;
```

```
    };

    \ref{\iotwod.absmatrix.nonmember}, non-members
    constexpr bool operator==(const abs_matrix& lhs, const abs_matrix& rhs)
      noexcept;
    constexpr bool operator!=(const abs_matrix& lhs, const abs_matrix& rhs)
      noexcept;
  }
}
```

9.3.5.2 `abs_matrix` constructors [io2d.absmatrix.cons]

```
constexpr abs_matrix() noexcept;
```

1 *Effects:* Equivalent to: `abs_matrix{ matrix_2d() };`

```
constexpr explicit abs_matrix(const matrix_2d& m) noexcept;
```

2 *Requires:* `m.is_invertible()` is `true`.

3 *Effects:* Constructs an object of type `abs_matrix`.

4 The transform matrix is `m`.

9.3.5.3 `abs_matrix` modifiers [io2d.absmatrix.modifiers]

```
constexpr void matrix(const matrix_2d& m) noexcept;
```

1 *Requires:* `m.is_invertible()` is `true`.

2 *Effects:* The transform matrix is `m`.

9.3.5.4 `abs_matrix` observers [io2d.absmatrix.observers]

```
constexpr matrix_2d matrix() const noexcept;
```

1 *Returns:* The transform matrix.

9.3.5.5 Non-member functions [io2d.absmatrix.nonmember]

```
constexpr bool operator==(const abs_matrix& lhs, const abs_matrix& rhs)
  noexcept;
```

1 *Returns:* `lhs.matrix() == rhs.matrix()`.

```
constexpr bool operator!=(const abs_matrix& lhs, const abs_matrix& rhs)
  noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.6 Class `rel_matrix` [io2d.relmatrix]

9.3.6.1 `rel_matrix` synopsis [io2d.relmatrix.synopsis]

1 The class `rel_matrix` describes a path item that is a path group instruction.

2 It has a transform matrix of type `matrix_2d`.

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class rel_matrix {
    public:
      // 9.3.6.2, construct:
      constexpr rel_matrix() noexcept;
```

```
      constexpr explicit rel_matrix(const matrix_2d& m) noexcept;

      // 9.3.6.3, modifiers:
      constexpr void matrix(const matrix_2d& m) noexcept;

      // 9.3.6.4, observers:
      constexpr matrix_2d matrix() const noexcept;
    };

    \ref{\iotwod.relmatrix.nonmember}, non-members
    constexpr bool operator==(const rel_matrix& lhs, const rel_matrix& rhs)
      noexcept;
    constexpr bool operator!=(const rel_matrix& lhs, const rel_matrix& rhs)
      noexcept;
  }
}
```

9.3.6.2 `rel_matrix` constructors [io2d.relmatrix.cons]

`constexpr rel_matrix() noexcept;`

1 *Effects:* Equivalent to: `rel_matrix{ matrix_2d() };`

`constexpr explicit rel_matrix(const matrix_2d& m) noexcept;`

2 *Requires:* `m.is_invertible()` is `true`.

3 *Effects:* Constructs an object of type `rel_matrix`.

4 The transform matrix is `m`.

9.3.6.3 `rel_matrix` modifiers [io2d.relmatrix.modifiers]

`constexpr void matrix(const matrix_2d& m) noexcept;`

1 *Requires:* `m.is_invertible()` is `true`.

2 *Effects:* The transform matrix is `m`.

9.3.6.4 `rel_matrix` observers [io2d.relmatrix.observers]

`constexpr matrix_2d matrix() const noexcept;`

1 *Returns:* The transform matrix.

9.3.6.5 Non-member functions [io2d.relmatrix.nonmember]

```
constexpr bool operator==(const rel_matrix& lhs, const rel_matrix& rhs)
  noexcept;
```

1 *Returns:* `lhs.matrix() == rhs.matrix()`.

```
constexpr bool operator!=(const rel_matrix& lhs, const rel_matrix& rhs)
  noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.7 Class `revert_matrix` [io2d.revertmatrix]

9.3.7.1 `revert_matrix` synopsis [io2d.revertmatrix.synopsis]

1 The class `revert_matrix` describes a path item that is a path group instruction.

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class revert_matrix {
    public:
      // 9.3.7.2, construct:
      constexpr revert_matrix() noexcept;
    };

    \ref{\iotwod.revertmatrix.nonmember}, non-members
    constexpr bool operator==(const revert_matrix& lhs,
      const revert_matrix& rhs) noexcept;
    constexpr bool operator!=(const revert_matrix& lhs,
      const revert_matrix& rhs) noexcept;
  }
}
```

9.3.7.2 `revert_matrix` constructors [io2d.revertmatrix.cons]

```
constexpr revert_matrix() noexcept;
```

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

9.3.7.3 Non-member functions [io2d.revertmatrix.nonmember]

```
constexpr bool operator==(const revert_matrix& lhs, const revert_matrix& rhs)
  noexcept;
```

1 *Returns:* `true`.

```
constexpr bool operator!=(const revert_matrix& lhs, const revert_matrix& rhs)
  noexcept;
```

2 *Returns:* `false`.

9.3.8 Class `abs_line` [io2d.absline]

1 The class `abs_line` describes a path item that is a path segment.

2 It has an *end point* of type `vector_2d`.

9.3.8.1 `abs_line` synopsis [io2d.absline.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class abs_line {
    public:
      // 9.3.8.2, construct:
      constexpr abs_line() noexcept;
      constexpr explicit abs_line(const vector_2d& pt) noexcept;

      // 9.3.8.3, modifiers:
      constexpr void to(const vector_2d& pt) noexcept;

      // 9.3.8.4, observers:
      constexpr vector_2d to() const noexcept;
    };

    \ref{\iotwod.absline.nonmember}, non-members
    constexpr bool operator==(const abs_line& lhs, const abs_line& rhs)
      noexcept;
```

```
      constexpr bool operator!=(const abs_line& lhs, const abs_line& rhs)
        noexcept;
    }
  }
```

9.3.8.2 abs_line constructors [io2d.absline.cons]

```
constexpr abs_line() noexcept;
```

1 *Effects:* Equivalent to: `abs_line{ vector_2d() };`

```
constexpr explicit abs_line(const vector_2d& pt) noexcept;
```

2 *Effects:* Constructs an object of type `abs_line`.

3 The end point is `pt`.

9.3.8.3 abs_line modifiers [io2d.absline.modifiers]

```
constexpr void to(const vector_2d& pt) noexcept;
```

1 *Effects:* The end point is `pt`.

9.3.8.4 abs_line observers [io2d.absline.observers]

```
constexpr vector_2d to() const noexcept;
```

1 *Returns:* The end point.

9.3.8.5 Non-member functions [io2d.absline.nonmember]

```
constexpr bool operator==(const abs_line& lhs, const abs_line& rhs) noexcept;
```

1 *Returns:* `lhs.to() == rhs.to()`.

```
constexpr bool operator!=(const abs_line& lhs, const abs_line& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.9 Class rel_line [io2d.relline]

1 The class `rel_line` describes a path item that is a path segment.

2 It has an *end point* of type `vector_2d`.

9.3.9.1 rel_line synopsis [io2d.relline.synopsis]

```
  namespace std::experimental::io2d::v1 {
    namespace path_data {
      class rel_line {
      public:
        // 9.3.9.2, construct:
        constexpr rel_line() noexcept;
        constexpr explicit rel_line(const vector_2d& pt) noexcept;

        // 9.3.9.3, modifiers:
        constexpr void to(const vector_2d& pt) noexcept;

        // 9.3.9.4, observers:
        constexpr vector_2d to() const noexcept;
      };

      \ref{\iotwod.relline.nonmember}, non-members
```

```
      constexpr bool operator==(const rel_line& lhs, const rel_line& rhs)
        noexcept;
      constexpr bool operator!=(const rel_line& lhs, const rel_line& rhs)
        noexcept;
    }
  }
```

9.3.9.2 `rel_line` constructors [io2d.relline.cons]

`constexpr rel_line() noexcept;`

1 *Effects:* Equivalent to: `rel_line{ vector_2d() };`

`constexpr explicit rel_line(const vector_2d& pt) noexcept;`

2 *Effects:* Constructs an object of type `rel_line`.

3 The end point is `pt`.

9.3.9.3 `rel_line` modifiers [io2d.relline.modifiers]

`constexpr void to(const vector_2d& pt) noexcept;`

1 *Effects:* The end point is `pt`.

9.3.9.4 `rel_line` observers [io2d.relline.observers]

`constexpr vector_2d to() const noexcept;`

1 *Returns:* The end point.

9.3.9.5 Non-member functions [io2d.relline.nonmember]

`constexpr bool operator==(const rel_line& lhs, const rel_line& rhs) noexcept;`

1 *Returns:* `lhs.to() == rhs.to()`.

`constexpr bool operator!=(const rel_line& lhs, const rel_line& rhs) noexcept;`

2 *Returns:* `!(lhs == rhs)`.

9.3.10 Class `abs_quadratic_curve` [io2d.absquadraticcurve]

1 The class `abs_quadratic_curve` describes a path item that is a path segment.

2 It has a *control point* of type `vector_2d` and an *end point* of type `vector_2d`.

9.3.10.1 `abs_quadratic_curve` synopsis [io2d.absquadraticcurve.synopsis]

```
  namespace std::experimental::io2d::v1 {
    namespace path_data {
      class abs_quadratic_curve {
      public:
        // 9.3.10.2, construct:
        constexpr abs_quadratic_curve() noexcept;
        constexpr abs_quadratic_curve(const vector_2d& cpt, const vector_2d& ept)
          noexcept;

        // 9.3.10.3, modifiers:
        constexpr void control(const vector_2d& cpt) noexcept;
        constexpr void end(const vector_2d& ept) noexcept;

        // 9.3.10.4, observers:
```

```
      constexpr vector_2d control() const noexcept;
      constexpr vector_2d end() const noexcept;
    };

    \ref{\iotwod.absquadraticcurve.nonmember}, non-members
    constexpr bool operator==(const abs_quadratic_curve& lhs,
      const abs_quadratic_curve& rhs) noexcept;
    constexpr bool operator!=(const abs_quadratic_curve& lhs,
      const abs_quadratic_curve& rhs) noexcept;
  }
}
```

9.3.10.2 `abs_quadratic_curve` constructors [io2d.absquadraticcurve.cons]

```
constexpr abs_quadratic_curve() noexcept;
```

1 *Effects:* Equivalent to: `abs_quadratic_curve{ vector_2d(), vector_2d() };`

```
constexpr abs_quadratic_curve(const vector_2d& cpt, const vector_2d& ept)
  noexcept;
```

2 *Effects:* Constructs an object of type `abs_quadratic_curve`.

3 The control point is `cpt`.

4 The end point is `ept`.

9.3.10.3 `abs_quadratic_curve` modifiers [io2d.absquadraticcurve.modifiers]

```
constexpr void control(const vector_2d& cpt) noexcept;
```

1 *Effects:* The control point is `cpt`.

```
constexpr void end(const vector_2d& ept) noexcept;
```

2 *Effects:* The end point is `ept`.

9.3.10.4 `abs_quadratic_curve` observers [io2d.absquadraticcurve.observers]

```
constexpr vector_2d control() const noexcept;
```

1 *Returns:* The control point.

```
constexpr vector_2d end() const noexcept;
```

2 *Returns:* The end point.

9.3.10.5 Non-member functions [io2d.absquadraticcurve.nonmember]

```
constexpr bool operator==(const abs_quadratic_curve& lhs,
  const abs_quadratic_curve& rhs) noexcept;
```

1 *Returns:* `lhs.control() == rhs.control() && lhs.end() == rhs.end()`.

```
constexpr bool operator!=(const abs_quadratic_curve& lhs,
  const abs_quadratic_curve& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.11 Class `rel_quadratic_curve` [io2d.relquadraticcurve]

1 The class `rel_quadratic_curve` describes a path item that is a path segment.

2 It has a *control point* of type `vector_2d` and an *end point* of type `vector_2d`.

9.3.11.1 rel_quadratic_curve synopsis [io2d.relquadraticcurve.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class rel_quadratic_curve {
    public:
      // 9.3.11.2, construct:
      constexpr rel_quadratic_curve() noexcept;
      constexpr rel_quadratic_curve(const vector_2d& cpt, const vector_2d& ept)
        noexcept;

      // 9.3.11.3, modifiers:
      constexpr void control(const vector_2d& cpt) noexcept;
      constexpr void end(const vector_2d& ept) noexcept;

      // 9.3.11.4, observers:
      constexpr vector_2d control() const noexcept;
      constexpr vector_2d end() const noexcept;
    };

    \ref{\iotwod.relquadraticcurve.nonmember}, non-members
    constexpr bool operator==(const rel_quadratic_curve& lhs,
      const rel_quadratic_curve& rhs) noexcept;
    constexpr bool operator!=(const rel_quadratic_curve& lhs,
      const rel_quadratic_curve& rhs) noexcept;
  }
}
```

9.3.11.2 rel_quadratic_curve constructors [io2d.relquadraticcurve.cons]

```
constexpr rel_quadratic_curve() noexcept;
```

1 *Effects:* Equivalent to: `rel_quadratic_curve{ vector_2d(), vector_2d() };`

```
constexpr rel_quadratic_curve(const vector_2d& cpt, const vector_2d& ept)
  noexcept;
```

2 *Effects:* Constructs an object of type `rel_quadratic_curve`.

3 The control point is `cpt`.

4 The end point is `ept`.

9.3.11.3 rel_quadratic_curve modifiers [io2d.relquadraticcurve.modifiers]

```
constexpr void control(const vector_2d& cpt) noexcept;
```

1 *Effects:* The control point is `cp`.

```
constexpr void end(const vector_2d& ept) noexcept;
```

2 *Effects:* The end point is `ept`.

9.3.11.4 rel_quadratic_curve observers [io2d.relquadraticcurve.observers]

```
constexpr vector_2d control() const noexcept;
```

1 *Returns:* The control point.

```
constexpr vector_2d end() const noexcept;
```

2 *Returns:* The end point.

9.3.11.5 Non-member functions [io2d.relquadraticcurve.nonmember]

```
constexpr bool operator==(const rel_quadratic_curve& lhs,
  const rel_quadratic_curve& rhs) noexcept;
```

1 *Returns:* `lhs.control() == rhs.control() && lhs.end() == rhs.end()`.

```
constexpr bool operator!=(const rel_quadratic_curve& lhs,
  const rel_quadratic_curve& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.12 Class `abs_cubic_curve` [io2d.abscubiccurve]

1 The class `abs_cubic_curve` describes a path item that is a path segment.

2 It has a *first control point* of type `vector_2d`, a *second control point* of type `vector_2d`, and an `end point` of type `vector_2d`.

9.3.12.1 `abs_cubic_curve` synopsis [io2d.abscubiccurve.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class abs_cubic_curve {
    public:
      // 9.3.12.2, construct:
      constexpr abs_cubic_curve() noexcept;
      constexpr abs_cubic_curve(const vector_2d& cpt1, const vector_2d& cpt2,
        const vector_2d& ept) noexcept;

      // 9.3.12.3, modifiers:
      constexpr void control_1(const vector_2d& cpt) noexcept;
      constexpr void control_2(const vector_2d& cpt) noexcept;
      constexpr void end(const vector_2d& ept) noexcept;

      // 9.3.12.4, observers:
      constexpr vector_2d control_1() const noexcept;
      constexpr vector_2d control_2() const noexcept;
      constexpr vector_2d end() const noexcept;
    };

    \ref{\iotwod.abscubiccurve.nonmember}, non-members
    constexpr bool operator==(const abs_cubic_curve& lhs,
      const abs_cubic_curve& rhs) noexcept;
    constexpr bool operator!=(const abs_cubic_curve& lhs,
      const abs_cubic_curve& rhs) noexcept;
  }
}
```

9.3.12.2 `abs_cubic_curve` constructors [io2d.abscubiccurve.cons]

```
constexpr abs_cubic_curve() noexcept;
```

1 *Effects:* Equivalent to `abs_cubic_curve{ vector_2d(), vector_2d(), vector_2d() }`.

```
constexpr abs_cubic_curve(const vector_2d& cpt1, const vector_2d& cpt2,
  const vector_2d& ept) noexcept;
```

2 *Effects:* Constructs an object of type `abs_cubic_curve`.

3 The first control point is `cpt1`.

4 The second control point is `cpt2`.

5 The end point is `ept`.

9.3.12.3 `abs_cubic_curve` modifiers [io2d.abscubiccurve.modifiers]

```
constexpr void control_1(const vector_2d& cpt) noexcept;
```

1 *Effects:* The first control point is `cpt`.

```
constexpr void control_2(const vector_2d& cpt) noexcept;
```

2 *Effects:* The second control point is `cpt`.

```
constexpr void end(const vector_2d& ept) noexcept;
```

3 *Effects:* The end point is `ept`.

9.3.12.4 `abs_cubic_curve` observers [io2d.abscubiccurve.observers]

```
constexpr vector_2d control_1() const noexcept;
```

1 *Returns:* The first control point.

```
constexpr vector_2d control_2() const noexcept;
```

2 *Returns:* The second control point.

```
constexpr vector_2d end() const noexcept;
```

3 *Returns:* The end point.

9.3.12.5 Non-member functions [io2d.abscubiccurve.nonmember]

```
constexpr bool operator==(const abs_cubic_curve& lhs,
  const abs_cubic_curve& rhs) noexcept;
```

1 *Returns:*

```
lhs.control_1() == rhs.control_1() && lhs.control_2() == rhs.control_2() &&
lhs.end() && rhs.end()
```

```
constexpr bool operator!=(const abs_cubic_curve& lhs, const abs_cubic_curve& rhs)
  noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.13 Class `rel_cubic_curve` [io2d.relcubiccurve]

1 The class `rel_cubic_curve` describes a path item that is a path segment.

2 It has a *first control point* of type `vector_2d`, a *second control point* of type `vector_2d`, and an `end point` of type `vector_2d`.

9.3.13.1 `rel_cubic_curve` synopsis [io2d.relcubiccurve.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class rel_cubic_curve {
    public:
      // 9.3.13.2, construct
      constexpr rel_cubic_curve() noexcept;
      constexpr rel_cubic_curve(const vector_2d& cpt1, const vector_2d& cpt2,
        const vector_2d& ept) noexcept;
```

```
    // 9.3.13.3, modifiers:
    constexpr void control_pt1(const vector_2d& cpt) noexcept;
    constexpr void control_pt2(const vector_2d& cpt) noexcept;
    constexpr void end_pt(const vector_2d& ept) noexcept;

    // 9.3.13.4, observers:
    constexpr vector_2d control_pt1() const noexcept;
    constexpr vector_2d control_pt2() const noexcept;
    constexpr vector_2d end_pt() const noexcept;
  };

  \ref{\iotwod.relcubiccurve.nonmember}, non-members
  constexpr bool operator==(const & lhs, const & rhs) noexcept;
  constexpr bool operator!=(const & lhs, const & rhs) noexcept;
 }
}
```

9.3.13.2 `rel_cubic_curve` constructors [io2d.relcubiccurve.cons]

```
constexpr rel_cubic_curve() noexcept;
```

1 *Effects:* Equivalent to `rel_cubic_curve{ vector_2d(), vector_2d(), vector_2d() }`

```
constexpr rel_cubic_curve(const vector_2d& cpt1, const vector_2d& cpt2,
  const vector_2d& ept) noexcept;
```

2 *Effects:* Constructs an object of type `rel_cubic_curve`.

3 The first control point is `cpt1`. The second control point is `cpt2`. The end point is `ept`.

9.3.13.3 `rel_cubic_curve` modifiers [io2d.relcubiccurve.modifiers]

```
constexpr void control_pt1(const vector_2d& cpt) noexcept;
```

1 *Effects:* The first control point is `cpt`.

```
constexpr void control_pt2(const vector_2d& cpt) noexcept;
```

2 *Effects:* The second control point is `cpt`.

```
constexpr void end_pt(const vector_2d& ept) noexcept;
```

3 *Effects:* The end point is `ept`.

9.3.13.4 `rel_cubic_curve` observers [io2d.relcubiccurve.observers]

```
constexpr vector_2d control_pt1() const noexcept;
```

1 *Returns:* The first control point.

```
constexpr vector_2d control_pt2() const noexcept;
```

2 *Returns:* The second control point.

```
constexpr vector_2d end_pt() const noexcept;
```

3 *Returns:* The end point.

9.3.13.5 Non-member functions [io2d.relcubiccurve.nonmember]

```
constexpr bool operator==(const rel_cubic_curve& lhs,
  const rel_cubic_curve& rhs) noexcept;
```

1 *Returns:*

```
lhs.control_1() == rhs.control_1() && lhs.control_2() == rhs.control_2() &&
lhs.end() && rhs.end()
```

```
constexpr bool operator!=(const rel_cubic_curve& lhs,
  const rel_cubic_curve& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.14 Class arc [io2d.arc]

9.3.14.1 In general [io2d.arc.general]

1 The class `arc` describes a path item that is a path segment.

2 It has a *radius* of type `vector_2d`, a *rotation* of type `float`, and a *start angle* of type `float`.

9.3.14.2 arc synopsis [io2d.arc.synopsis]

```
namespace std::experimental::io2d::v1 {
  namespace path_data {
    class arc {
    public:
      // 9.3.14.3, construct/copy/move/destroy:
      constexpr arc() noexcept;
      constexpr arc(const vector_2d& rad,
        float rot, float sang) noexcept;

      // 9.3.14.4, modifiers:
      constexpr void radius(const vector_2d& rad) noexcept;
      constexpr void rotation(float rot) noexcept;
      constexpr void start_angle(float radians) noexcept;

      // 9.3.14.5, observers:
      constexpr vector_2d radius() const noexcept;
      constexpr float rotation() const noexcept;
      constexpr float start_angle() const noexcept;
      vector_2d center(const vector_2d& cpt, const matrix_2d& m = matrix_2d{})
        const noexcept;
      vector_2d end_pt(const vector_2d& cpt, const matrix_2d& m = matrix_2d{})
        const noexcept;
    };

    // 9.3.14.6, non-members
    constexpr bool operator==(const arc& lhs, const arc& rhs) noexcept;
    constexpr bool operator!=(const arc& lhs, const arc& rhs) noexcept;
  }
}
```

9.3.14.3 arc constructors [io2d.arc.cons]

```
constexpr arc() noexcept;
```

1 *Effects:* Equivalent to: `arc{ vector_2d(10.0f, 10.0f), pi<float>, pi<float> };`.

```
constexpr arc(const vector_2d& rad, float rot,
  float start_angle = pi<float>) noexcept;
```

2 *Effects:* Constructs an object of type `arc`.

3 The radius is `rad`.

4 The rotation is `rot`.

5 The start angle is `sang`.

9.3.14.4 arc modifiers [io2d.arc.modifiers]

```
constexpr void radius(const vector_2d& rad) noexcept;
```

1 *Effects:* The radius is `rad`.

```
constexpr void rotation(float rot) noexcept;
```

2 *Effects:* The rotation is `rot`.

```
constexpr void start_angle(float sang) noexcept;
```

3 *Effects:* The start angle is `sang`.

9.3.14.5 arc observers [io2d.arc.observers]

```
constexpr vector_2d radius() const noexcept;
```

1 *Returns:* The radius.

```
constexpr float rotation() const noexcept;
```

2 *Returns:* The rotation.

```
constexpr float start_angle() const noexcept;
```

3 *Returns:* The start angle.

```
vector_2d center(const vector_2d& cpt, const matrix_2d& m = matrix_2d{})
  const noexcept;
```

4 *Returns:* As-if:

```
auto lmtx = m;
lmtx.m20(0.0f);
lmtx.m21(0.0f);
auto centerOffset = point_for_angle(two_pi<float> - _Start_angle, _Radius);
centerOffset.y(-centerOffset.y());
return cpt - centerOffset * lmtx;
```

```
vector_2d end_pt(const vector_2d& cpt, const matrix_2d& m = matrix_2d{})
  const noexcept;
```

5 *Returns:* As-if:

```
auto lmtx = m;
auto tfrm = matrix_2d::init_rotate(_Start_angle + _Rotation);
lmtx.m20(0.0f);
lmtx.m21(0.0f);
auto pt = (_Radius * tfrm);
pt.y(-pt.y());
return cpt + pt * lmtx;
```

9.3.14.6 Non-member functions [io2d.arc.nonmember]

```
constexpr bool operator==(const arc& lhs, const arc& rhs) noexcept;
```

1 *Returns:*

```
lhs.radius() == rhs.radius() && lhs.rotation() == rhs.rotation() &&
lhs.start_angle() && rhs.start_angle()
```

```
constexpr bool operator!=(const arc& lhs, const arc& rhs) noexcept;
```

2 *Returns:* `!(lhs == rhs)`.

9.3.15 Path group interpretation [io2d.paths.interpretation]

1 This subclause describes how to interpret a path group for use in a rendering and composing operation.

2 Interpreting a path group consists of sequentially evaluating the `path_data::path_item` objects in a path group and transforming them into zero or more paths as-if in the manner specified in this subclause.

3 The interpretation of a path group requires the state data specified in Table 2.

Table 2 — Path group interpretation state data

Name	**Description**	**Type**	**Initial value**
`mtx`	Path group transformation matrix	`matrix_2d`	`matrix_2d{ }`
`currPt`	Current point	`vector_2d`	*unspecified*
`lnPt`	Last new point	`vector_2d`	*unspecified*
`mtxStk`	Matrix stack	`stack<matrix_2d>`	`stack<matrix_2d>{ }`

4 When interpreting a path group, until a `path_data::abs_new_path` path item is reached, a path shall only contain path group instruction path items; no diagnostic is required. If a path is a degenerate path, none of its path items have any effects, with two exceptions:

(4.1) — the path's `path_data::abs_new_path` or `path_data::rel_new_path` path item sets the value of `currPt` as-if the path item was interpreted; and,

(4.2) — any path group instruction path items are evaluated with full effect.

.

5 The effects of a path item contained in a `path_data::path_item` object when that object is being evaluated during path group interpretation are described in Table 3.

Table 3 — Path item interpretation effects

Path item	**Effects**
`path_data::abs_new_path p`	Creates a new path. Sets `currPt` to `mtx.transform_point({ 0.0f, 0.0f }) + p.at()`. Sets `lnPt` to `currPt`.
`path_data::rel_new_path p`	Let `mm` equal `mtx`. Evaluate `mm.m20(0.0f)` and `mm.m21(0.0f)`. Creates a new path. Sets `currPt` to `currPt + p.at() * mm`. Sets `lnPt` to `currPt`.
`path_data::close_path p`	Creates a line from `currPt` to `lnPt`. Makes the current path a closed path. Creates a new path. Sets `currPt` to `lnPt`.
`path_data::abs_matrix p`	Calls `mtxStk.push(mtx)`. Sets `mtx` to `p.matrix()`.
`path_data::rel_matrix p`	Calls `mtxStk.push(mtx)`. Sets `mtx` to `mtx * p.matrix()`.
`path_data::revert_matrix p`	If `mtxStk.empty()` is `false`, sets `mtx` to `ognStk.top()` then calls `ognStk.pop()`. Otherwise sets `mtx` to its initial value as specified in Table 2.
`path_data::abs_line p`	Let `pt` equal `mtx.transform_point(p.to() - currPt) + currPt`. Creates a line from `currPt` to `pt`. Sets `currPt` to `pt`.
`path_data::rel_line p`	Let `mm` equal `mtx`. Evaluate `mm.m20(0.0f)` and `mm.m21(0.0f)`. Let `pt` equal `currPt + p.to() * mm`. Creates a line from `currPt` to `pt`. Sets `currPt` to `pt`.

Table 3 — Path item interpretation effects (continued)

Path item	Effects
path_data::abs_quadratic_curve p	Let cpt equal mtx.transform_point(p.control_pt() - currPt) + currPt. Let ept equal mtx.transform_point(p.end_pt() - currPt) + currPt. Creates a quadratic Bézier curve from currPt to ept using cpt as the curve's control point. Sets currPt to ept.
path_data::rel_quadratic_curve p	Let mm equal mtx. Evaluate mm.m20(0.0f) and mm.m21(0.0f). Let cpt equal currPt + p.control_pt() * mm. Let ept equal currPt + p.control_pt() * mm + p.end_pt() * mm. Creates a quadratic Bézier curve from currPt to ept using cpt as the curve's control point. Sets currPt to ept.
path_data::abs_cubic_curve p	Let cpt1 equal mtx.transform_point(p.control_pt1() - currPt) + currPt. Let cpt2 equal mtx.transform_point(p.control_pt2() - currPt) + currPt. Let ept equal mtx.transform_point(p.end_pt() - currPt) + currPt. Creates a cubic Bézier curve from currPt to ept using cpt1 as the curve's first control point and cpt2 as the curve's second control point. Sets currPt to ept.
path_data::rel_cubic_curve p	Let mm equal mtx. Evaluate mm.m20(0.0f) and mm.m21(0.0f). Let cpt1 equal currPt + p.control_pt1() * mm. Let cpt2 equal currPt + p.control_pt1() * mm + p.control_pt2() * mm. Let ept equal currPt + p.control_pt1() * mm + p.control_pt2() * mm + p.end_pt() * mm. Creates a cubic Bézier curve from currPt to ept using cpt1 as the curve's first control point and cpt2 as the curve's second control point. Sets currPt to ept.
path_data::arc p	Let mm equal mtx. Evaluate mm.m20(0.0f) and mm.m21(0.0f). Creates an arc. It begins at currPt, which is at p.start_angle() radians on the arc and rotates p.rotation() radians. If p.rotation() is positive, rotation is counterclockwise, otherwise it is clockwise. The center of the arc is located at p.center(currPt, mm). The arc ends at p.end_pt(currPt, mm). Sets currPt to p.end_pt(currPt, mm). [*Note:* p.radius(), which specifies the radius of the arc, is implicitly included in the above statement of effects by the specifications of the center of the arc and the end of the arcs. The use of the current point as the origin for the application of the path group transformation matrix is also implicitly included by the same specifications. — *end note*]

9.4 Class path_group [io2d.pathgroup]

1 The class path_group contains the data that result from interpreting 9.3.15 a sequence of path_data::path_item objects.

2 A path_group object is used by most rendering and composing operations.

9.4.1 path_group synopsis [io2d.pathgroup.synopsis]

```
namespace std::experimental::io2d::v1 {
  class path_group {
    public:
    // 9.4.2, construct/copy/destroy:
    explicit path_group(const path_builder& pb);
    template <class ForwardIterator>
    path_group(ForwardIterator first, ForwardIterator last);
```

```
    };
  }
```

9.4.2 path_group constructors [io2d.pathgroup.cons]

```
explicit path_group(const path_builder& pb);
```

1 *Effects:* Equivalent to: `path_group{ begin(pb), end(pb) };`.

```
template <class ForwardIterator>
path_group(ForwardIterator first, ForwardIterator last);
```

2 *Effects:* Constructs an object of type `path_group`.

9.5 Class path_builder [io2d.pathbuilder]

1 The class `path_builder` is a container that stores and manipulates objects of type `path_data::path_item` from which `path_group` objects are created.

2 A `path_builder` is a contiguous container. (See [container.requirements.general] in N4618.)

3 The collection of `path_data::path_item` objects in a path builder is referred to as its path group.

9.5.1 path_builder synopsis [io2d.pathbuilder.synopsis]

```
  namespace std::experimental::io2d::v1 {
    template <class Allocator = allocator<path_data::path_item>>
    class path_builder {
    public:
      using value_type = path_data::path_item;
      using allocator_type = Allocator;
      using reference = value_type&;
      using const_reference = const value_type&;
      using size_type       = implementation-defined. // See [container.requirements] in N4618.
      using difference_type = implementation-defined. // See [container.requirements] in N4618.
      using iterator       = implementation-defined. // See [container.requirements] in N4618.
      using const_iterator = implementation-defined. // See [container.requirements] in N4618.
      using reverse_iterator       = std::reverse_iterator<iterator>;
      using const_reverse_iterator = std::reverse_iterator<const_iterator>;

      // 9.5.3, construct, copy, move, destroy:
      path_builder() noexcept(noexcept(Allocator())) :
        path_builder(Allocator()) { }
      explicit path_builder(const Allocator&) noexcept;
      explicit path_builder(size_type n, const Allocator& = Allocator());
      path_builder(size_type n, const value_type& value,
        const Allocator& = Allocator());
      template <class InputIterator>
      path_builder(InputIterator first, InputIterator last,
        const Allocator& = Allocator());
      path_builder(const path_builder& x);
      path_builder(path_builder&&) noexcept;
      path_builder(const path_builder&, const Allocator&);
      path_builder(path_builder&&, const Allocator&);
      path_builder(initializer_list<value_type>, const Allocator& = Allocator());
      ~path_builder();
      path_builder& operator=(const path_builder& x);
      path_builder& operator=(path_builder&& x)
        noexcept(
```

```
    allocator_traits<Allocator>::propagate_on_container_move_assignment::value
    ||
    allocator_traits<Allocator>::is_always_equal::value);
  path_builder& operator=(initializer_list<value_type>);
  template <class InputIterator>
  void assign(InputIterator first, InputIterator last);
  void assign(size_type n, const value_type& u);
  void assign(initializer_list<value_type>);
  allocator_type get_allocator() const noexcept;

  // 9.5.6, iterators:
  iterator begin() noexcept;
  const_iterator begin() const noexcept;
  const_iterator cbegin() const noexcept;

  iterator end() noexcept;
  const_iterator end() const noexcept;
  const_iterator cend() const noexcept;

  reverse_iterator rbegin() noexcept;
  const_reverse_iterator rbegin() const noexcept;
  const_reverse_iterator crbegin() const noexcept;

  reverse_iterator rend() noexcept;
  const_reverse_iterator rend() const noexcept;
  const_reverse_iterator crend() const noexcept;

  // 9.5.4, capacity
  bool empty() const noexcept;
  size_type size() const noexcept;
  size_type max_size() const noexcept;
  size_type capacity() const noexcept;
  void resize(size_type sz);
  void resize(size_type sz, const value_type& c);
  void reserve(size_type n);
  void shrink_to_fit();

  // element access:
  reference operator[](size_type n);
  const_reference operator[](size_type n) const;
  const_reference at(size_type n) const;
  reference at(size_type n);
  reference front();
  const_reference front() const;
  reference back();
  const_reference back() const;

  // 9.5.5, modifiers:
  void new_path(const vector_2d& pt) noexcept;
  void rel_new_path(const vector_2d& pt) noexcept;
  void close_path() noexcept;
  void matrix(const matrix_2d& m) noexcept;
  void rel_matrix(const matrix_2d& m) noexcept;
  void revert_matrix() noexcept;
  void line(const vector_2d& pt) noexcept;
```

```
    void rel_line(const vector_2d& dpt) noexcept;
    void quadratic_curve(const vector_2d& pt0, const vector_2d& pt2)
      noexcept;
    void rel_quadratic_curve(const vector_2d& pt0, const vector_2d& pt2)
      noexcept;
    void cubic_curve(const vector_2d& pt0, const vector_2d& pt1,
      const vector_2d& pt2) noexcept;
    void rel_cubic_curve(const vector_2d& dpt0, const vector_2d& dpt1,
      const vector_2d& dpt2) noexcept;
    void arc(const vector_2d& rad, float rot, float sang = pi<float>)
      noexcept;

    template <class... Args>
    reference emplace_back(Args&&... args);
    void push_back(const value_type& x);
    void push_back(value_type&& x);
    void pop_back();
    template <class... Args>
    iterator emplace(const_iterator position, Args&&... args);
    iterator insert(const_iterator position, const value_type& x);
    iterator insert(const_iterator position, value_type&& x);
    iterator insert(const_iterator position, size_type n, const value_type& x);
    template <class InputIterator>
    iterator insert(const_iterator position, InputIterator first,
      InputIterator last);
    iterator insert(const_iterator position,
      initializer_list<value_type> il);
    iterator erase(const_iterator position);
    iterator erase(const_iterator first, const_iterator last);
    void swap(path_builder&)
      noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value
        || allocator_traits<Allocator>::is_always_equal::value);
    void clear() noexcept;
  };

  template <class Allocator>
  bool operator==(const path_builder<Allocator>& lhs,
    const path_builder<Allocator>& rhs);
  template <class Allocator>
  bool operator!=(const path_builder<Allocator>& lhs,
    const path_builder<Allocator>& rhs);

  // 9.5.7, specialized algorithms:
  template <class Allocator>
  void swap(path_builder<Allocator>& lhs, path_builder<Allocator>& rhs)
    noexcept(noexcept(lhs.swap(rhs)));
}
```

9.5.2 path_builder container requirements [io2d.pathbuilder.containerrequirements]

1 This class is a sequence container, as defined in [containers] in N4618, and all sequence container requirements that apply specifically to `vector` shall also apply to this class.

9.5.3 path_builder constructors, copy, and assignment [io2d.pathbuilder.cons]

```
explicit path_builder(const Allocator&);
```

1 *Effects:* Constructs an empty `path_builder`, using the specified allocator.

2 *Complexity:* Constant.

```
explicit path_builder(size_type n, const Allocator& = Allocator());
```

3 *Effects:* Constructs a `path_builder` with `n` default-inserted elements using the specified allocator.

4 *Complexity:* Linear in `n`.

```
path_builder(size_type n, const value_type& value,
  const Allocator& = Allocator());
```

5 *Requires:* `value_type` shall be `CopyInsertable` into `*this`.

6 *Effects:* Constructs a `path_builder` with n copies of `value`, using the specified allocator.

7 *Complexity:* Linear in `n`.

```
template <class InputIterator>
path_builder(InputIterator first, InputIterator last,
  const Allocator& = Allocator());
```

8 *Effects:* Constructs a `path_builder` equal to the range `[first, last)`, using the specified allocator.

9 *Complexity:* Makes only N calls to the copy constructor of `value_type` (where N is the distance between `first` and `last`) and no reallocations if iterators `first` and `last` are of forward, bidirectional, or random access categories. It makes order `N` calls to the copy constructor of `value_type` and order $\log(N)$ reallocations if they are just input iterators.

9.5.4 `path_builder` capacity [io2d.pathbuilder.capacity]

```
size_type capacity() const noexcept;
```

1 *Returns:* The total number of elements that the path builder can hold without requiring reallocation.

```
void reserve(size_type n);
```

2 *Requires:* `value_type` shall be `MoveInsertable` into `*this`.

3 *Effects:* A directive that informs a path builder of a planned change in size, so that it can manage the storage allocation accordingly. After `reserve()`, `capacity()` is greater or equal to the argument of `reserve` if reallocation happens; and equal to the previous value of `capacity()` otherwise. Reallocation happens at this point if and only if the current capacity is less than the argument of `reserve()`. If an exception is thrown other than by the move constructor of a non-`CopyInsertable` type, there are no effects.

4 *Complexity:* It does not change the size of the sequence and takes at most linear time in the size of the sequence.

5 *Throws:* `length_error` if `n > max_size()`.[1]

6 *Remarks:* Reallocation invalidates all the references, pointers, and iterators referring to the elements in the sequence. No reallocation shall take place during insertions that happen after a call to `reserve()` until the time when an insertion would make the size of the vector greater than the value of `capacity()`.

```
void shrink_to_fit();
```

7 *Requires:* `value_type` shall be `MoveInsertable` into `*this`.

8 *Effects:* `shrink_to_fit` is a non-binding request to reduce `capacity()` to `size()`. [*Note:* The request is non-binding to allow latitude for implementation-specific optimizations. — *end note*] It does not

1) `reserve()` uses `Allocator::allocate()` which may throw an appropriate exception.

increase `capacity()`, but may reduce `capacity()` by causing reallocation. If an exception is thrown other than by the move constructor of a non-`CopyInsertable` `value_type` there are no effects.

9 *Complexity:* Linear in the size of the sequence.

10 *Remarks:* Reallocation invalidates all the references, pointers, and iterators referring to the elements in the sequence. If no reallocation happens, they remain valid.

```
void swap(path_builder&)
  noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value ||
  allocator_traits<Allocator>::is_always_equal::value);
```

11 *Effects:* Exchanges the contents and `capacity()` of `*this` with that of `x`.

12 *Complexity:* Constant time.

resize

```
void resize(size_type sz);
```

13 *Effects:* If `sz < size()`, erases the last `size() - sz` elements from the sequence. Otherwise, appends `sz - size()` default-inserted elements to the sequence.

14 *Requires:* `value_type` shall be `MoveInsertable` and `DefaultInsertable` into `*this`.

15 *Remarks:* If an exception is thrown other than by the move constructor of a non-`CopyInsertable` `value_type` there are no effects.

resize

```
void resize(size_type sz, const value_type& c);
```

16 *Effects:* If `sz < size()`, erases the last `size() - sz` elements from the sequence. Otherwise, appends `sz - size()` copies of `c` to the sequence.

17 *Requires:* `value_type` shall be `CopyInsertable` into `*this`.

18 *Remarks:* If an exception is thrown there are no effects.

9.5.5 path_builder modifiers [io2d.pathbuilder.modifiers]

```
void new_path(const vector_2d& pt) noexcept;
```

1 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::abs_new_path(pt)` to the end of the path group.

```
void rel_new_path(const vector_2d& pt) noexcept;
```

2 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::rel_new_path(pt)` to the end of the path group.

```
void close_path() noexcept;
```

3 *Requires:* The current point contains a value.

4 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::close_path()` to the end of the path group.

```
void matrix(const matrix_2d& m) noexcept;
```

5 *Requires:* The matrix `m` shall be invertible.

6 *Effects:* Adds a `path_data::path_item` object constructed from `(path_data::abs_matrix(m)` to the end of the path group.

```
void rel_matrix(const matrix_2d& m) noexcept;
```

7 *Requires:* The matrix `m` shall be invertible.

8 *Effects:* Adds a `path_data::path_item` object constructed from `(path_data::rel_matrix(m)` to the end of the path group.

```
void revert_matrix() noexcept;
```

9 *Effects:* Adds a `path_data::path_item` object constructed from `(path_data::revert_matrix()` to the end of the path group.

```
void line(const vector_2d& pt) noexcept;
```

10 Adds a `path_data::path_item` object constructed from `path_data::abs_line(pt)` to the end of the path group.

```
void rel_line(const vector_2d& dpt) noexcept;
```

11 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::rel_line(pt)` to the end of the path group.

```
void quadratic_curve(const vector_2d& pt0, const vector_2d& pt1) noexcept;
```

12 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::abs_quadratic_curve(pt0, pt1)` to the end of the path group.

```
void rel_quadratic_curve(const vector_2d& dpt0, const vector_2d& dpt1)
  noexcept;
```

13 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::rel_quadratic_curve(dpt0, dpt1)` to the end of the path group.

```
void cubic_curve(const vector_2d& pt0, const vector_2d& pt1,
  const vector_2d& pt2) noexcept;
```

14 15 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::abs_cubic_curve(pt0, pt1, pt2)` to the end of the path group.

```
void rel_cubic_curve(const vector_2d& dpt0, const vector_2d& dpt1,
  const vector_2d& dpt2) noexcept;
```

16 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::rel_cubic_curve(dpt0, dpt1, dpt2)` to the end of the path group.

```
void arc(const vector_2d& rad, float rot, float sang) noexcept;
```

17 *Effects:* Adds a `path_data::path_item` object constructed from `path_data::arc(rad, rot, sang)` to the end of the path group.

```
iterator insert(const_iterator position, const value_type& x);
iterator insert(const_iterator position, value_type&& x);
iterator insert(const_iterator position, size_type n, const value_type& x);
template <class InputIterator>
iterator insert(const_iterator position, InputIterator first,
  InputIterator last);
iterator insert(const_iterator position, initializer_list<value_type>);
template <class... Args>
reference emplace_back(Args&&... args);
template <class... Args>
iterator emplace(const_iterator position, Args&&... args);
```

```
void push_back(const value_type& x);
void push_back(value_type&& x);
```

18 *Remarks:* Causes reallocation if the new size is greater than the old capacity. Reallocation invalidates all the references, pointers, and iterators referring to the elements in the sequence. If no reallocation happens, all the iterators and references before the insertion point remain valid. If an exception is thrown other than by the copy constructor, move constructor, assignment operator, or move assignment operator of `value_type` or by any `InputIterator` operation there are no effects. If an exception is thrown while inserting a single element at the end and `value_type` is `CopyInsertable` or `is_nothrow_move_constructible_v<value_type>` is `true`, there are no effects. Otherwise, if an exception is thrown by the move constructor of a non-`CopyInsertable` `value_type`, the effects are unspecified.

19 *Complexity:* The complexity is linear in the number of elements inserted plus the distance to the end of the path builder.

```
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
void pop_back();
```

20 *Effects:* Invalidates iterators and references at or after the point of the erase.

21 *Complexity:* The destructor of `value_type` is called the number of times equal to the number of the elements erased, but the assignment operator of `value_type` is called the number of times equal to the number of elements in the path builder after the erased elements.

22 *Throws:* Nothing unless an exception is thrown by the copy constructor, move constructor, assignment operator, or move assignment operator of `value_type`.

9.5.6 path_builder iterators [io2d.pathbuilder.iterators]

```
iterator begin() noexcept;
const_iterator begin() const noexcept;
const_iterator cbegin() const noexcept;
```

1 *Returns:* An iterator referring to the first `path_data::path_item` item in the path group.

2 *Remarks:* Changing a `path_data::path_item` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_item` object or of any subsequent `path_data::path_item` object in the path group produces undefined behavior when the path group is interpreted as described in 9.3.15 unless all of the violations are fixed prior to such interpretation.

```
iterator end() noexcept;
const_iterator end() const noexcept;
const_iterator cend() const noexcept;
```

3 *Returns:* An iterator which is the past-the-end value.

4 *Remarks:* Changing a `path_data::path_item` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_item` object or of any subsequent `path_data::path_item` object in the path group produces undefined behavior when the path group is interpreted as described in 9.3.15 unless all of the violations are fixed prior to such interpretation.

```
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
const_reverse_iterator crbegin() const noexcept;
```

5 *Returns:* An iterator which is semantically equivalent to `reverse_iterator(end)`.

6 *Remarks:* Changing a `path_data::path_item` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_item` object or of any subsequent `path_-`

data::path_item object in the path group produces undefined behavior when the path group is interpreted as described in 9.3.15 all of the violations are fixed prior to such interpretation.

```
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
const_reverse_iterator crend() const noexcept;
```

7 *Returns:* An iterator which is semantically equivalent to `reverse_iterator(begin)`.

8 *Remarks:* Changing a `path_data::path_item` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_item` object or of any subsequent `path_data::path_item` object in the path group produces undefined behavior when the path group is interpreted as described in 9.3.15 unless all of the violations are fixed prior to such interpretation.

9.5.7 path_builder specialized algorithms [io2d.pathbuilder.special]

```
template <class Allocator>
void swap(path_builder<Allocator>& lhs, path_builder<Allocator>& rhs)
  noexcept(noexcept(lhs.swap(rhs)));
```

1 *Effects:* As if by `lhs.swap(rhs)`.

10 Brushes [io2d.brushes]

10.1 Overview of brushes [io2d.brushes.intro]

1 Brushes contain visual data and serve as sources of visual data for rendering and composing operations.

2 There are four types of brushes:

(2.1) — solid color;

(2.2) — linear gradient;

(2.3) — radial gradient; and,

(2.4) — surface.

3 Once a brush is created, its visual data is immutable.

4 [*Note:* While copy and move operations along with a swap operation can change the visual data that a brush contains, the visual data itself is not modified. — *end note*]

5 A brush is used either as a *source brush* or a *mask brush* (11.15.3.2).

6 When a brush is used in a rendering and composing operation, if it is used as a source brush, it has a `brush_props` object that describes how the brush is interpreted for purposes of sampling. If it is used as a mask brush, it has a `mask_props` object that describes how the brush is interpreted for purposes of sampling.

7 The `brush_props` (11.11.1) and `mask_props` (11.14.1) classes each have a *wrap mode* and a *filter*. The `brush_props` class also has a *brush matrix* and a *fill rule*. The `mask_props` class also has a `mask matrix`. Where possible, the terms that are common between the two classes are referenced without regard to whether the brush is being used as a source brush or a mask brush.

8 Solid color brushes are unbounded and as such always produce the same visual data when sampled from, regardless of the requested point.

9 Linear gradient and radial gradient brushes share similarities with each other that are not shared by the other types of brushes. This is discussed in more detail elsewhere (10.2).

10 Surface brushes are constructed from an `image_surface` object. Their visual data is a pixmap, which has implications on sampling from the brush that are not present in the other brush types.

10.2 Gradient brushes [io2d.gradients]

10.2.1 Common properties of gradients [io2d.gradients.common]

1 Gradients are formed, in part, from a collection of `color_stop` objects.

2 The collection of `color_stop` objects contribute to defining a brush which, when sampled from, returns a value that is interpolated based on those color stops.

10.2.2 Linear gradients [io2d.gradients.linear]

1 A linear gradient is a type of gradient.

2 A linear gradient has a *begin point* and an *end point*, each of which are objects of type `vector_2d`.

3 A linear gradient for which the distance between its begin point and its end point is not greater than `numeric_limits<float>::epsilon()` is a *degenerate linear gradient*.

4 All attempts to sample from a a degenerate linear gradient return the color `rgba_color::transparent_black()`. The remainder of 10.2 is inapplicable to degenerate linear gradients.

5 The begin point and end point of a linear gradient define a line segment, with a color stop offset value of 0.0f corresponding to the begin point and a color stop offset value of 1.0f corresponding to the end point.

6 Color stop offset values in the range `[0.0f, 1.0f]` linearly correspond to points on the line segment.

7 [*Example:* Given a linear gradient with a begin point of `vector_2d(0.0f, 0.0f)` and an end point of `vector_2d(10.0f, 5.0f)`, a color stop offset value of 0.6f would correspond to the point `vector_2d(6.0f, 3.0f)`. — *end example*]

8 To determine the offset value of a point p for a linear gradient, perform the following steps:

a) Create a line at the begin point of the linear gradient, the *begin line*, and another line at the end point of the linear gradient, the *end line*, with each line being perpendicular to the *gradient line segment*, which is the line segment delineated by the begin point and the end point.

b) Using the begin line, p, and the end line, create a line, the p *line*, which is parallel to the gradient line segment.

c) Defining dp as the distance between p and the point where the p line intersects the begin line and dt as the distance between the point where the p line intersects the begin line and the point where the p line intersects the end line, the offset value of p is $dp \div dt$.

d) The offset value shall be negative if

(8.1) — p is not on the line segment delineated by the point where the p line intersects the begin line and the point where the p line intersects the end line; and,

(8.2) — the distance between p and the point where the p line intersects the begin line is less than the distance between p and the point where the p line intersects the end line.

10.2.3 Radial gradients [io2d.gradients.radial]

1 A radial gradient is a type of gradient.

2 Aa radial gradient has a *start circle* and an *end circle*, each of which is defined by a `circle` object.

3 A radial gradient is a *degenerate radial gradient* if:

(3.1) — its start circle has a negative radius; or,

(3.2) — its end circle has a negative radius; or,

(3.3) — the distance between the center point of its start circle and the center point of its end circle is not greater than `numeric_limits<float>::epsilon()` and the difference between the radius of its start circle and the radius of its end circle is not greater than `numeric_limits<float>::epsilon()`; or,

(3.4) — its start circle has a radius of 0.0f and its end circle has a radius of 0.0f.

4 All attempts to sample from a `brush` object created using a degenerate radial gradient return the color `rgba_color::transparent_black()`. The remainder of 10.2 is inapplicable to degenerate radial gradients.

5 A color stop offset of 0.0f corresponds to all points along the diameter of the start circle or to its center point if it has a radius value of 0.0f.

6 A color stop offset of 1.0f corresponds to all points along the diameter of the end circle or to its center point if it has a radius value of 0.0f.

7 A radial gradient shall be rendered as a continuous series of interpolated circles defined by the following equations:

a) $x(o) = x_{start} + o \times (x_{end} - x_{start})$

b) $y(o) = y_{start} + o \times (y_{end} - y_{start})$

c) $radius(o) = radius_{start} + o \times (radius_{end} - radius_{start})$

where o is a color stop offset value.

8 The range of potential values for o shall be determined by the *wrap mode* (10.1):

(8.1) — For `wrap_mode::none`, the range of potential values for o is `[0, 1]`.

(8.2) — For all other `wrap_mode` values, the range of potential values for o is [`numeric_limits<float>::lowest(),numeric_limits<float>::max()`].

9 The interpolated circles shall be rendered starting from the smallest potential value of o.

10 An interpolated circle shall not be rendered if its value for o results in $radius(o)$ evaluating to a negative value.

10.2.4 Sampling from gradients [io2d.gradients.sampling]

1 For any offset value o, its color value shall be determined according to the following rules:

a) If there are less than two color stops or if all color stops have the same offset value, then the color value of every offset value shall be `rgba_color::transparent_black()` and the remainder of these rules are inapplicable.

b) If exactly one color stop has an offset value equal to o, o's color value shall be the color value of that color stop and the remainder of these rules are inapplicable.

c) If two or more color stops have an offset value equal to o, o's color value shall be the color value of the color stop which has the lowest index value among the set of color stops that have an offset value equal to o and the remainder of 10.2.4 is inapplicable.

d) When no color stop has the offset value of `0.0f`, then, defining n to be the offset value that is nearest to `0.0f` among the offset values in the set of all color stops, if o is in the offset range $[0, n)$, o's color value shall be `rgba_color::transparent_black()` and the remainder of these rules are inapplicable. [*Note:* Since the range described does not include n, it does not matter how many color stops have n as their offset value for purposes of this rule. — *end note*]

e) When no color stop has the offset value of `1.0f`, then, defining n to be the offset value that is nearest to `1.0f` among the offset values in the set of all color stops, if o is in the offset range $(n, 1]$, o's color value shall be `rgba_color::transparent_black()` and the remainder of these rules are inapplicable. [*Note:* Since the range described does not include n, it does not matter how many color stops have n as their offset value for purposes of this rule. — *end note*]

f) Each color stop has, at most, two adjacent color stops: one to its left and one to its right.

g) Adjacency of color stops is initially determined by offset values. If two or more color stops have the same offset value then index values are used to determine adjacency as described below.

h) For each color stop a, the *set of color stops to its left* are those color stops which have an offset value which is closer to `0.0f` than a's offset value. [*Note:* This includes any color stops with an offset value of `0.0f` provided that a's offset value is not `0.0f`. — *end note*]

i) For each color stop b, the *set of color stops to its right* are those color stops which have an offset value which is closer to `1.0f` than b's offset value. [*Note:* This includes any color stops with an offset value of `1.0f` provided that b's offset value is not `1.0f`. — *end note*]

j) A color stop which has an offset value of `0.0f` does not have an adjacent color stop to its left.

k) A color stop which has an offset value of `1.0f` does not have an adjacent color stop to its right.

l) If a color stop a's set of color stops to its left consists of exactly one color stop, that color stop is the color stop that is adjacent to a on its left.

m) If a color stop b's set of color stops to its right consists of exactly one color stop, that color stop is the color stop that is adjacent to b on its right.

n) If two or more color stops have the same offset value then the color stop with the lowest index value is the only color stop from that set of color stops which can have a color stop that is adjacent to it on its left and the color stop with the highest index value is the only color stop from that set of color stops which can have a color stop that is adjacent to it on its right. This rule takes precedence over all of the remaining rules.

o) If a color stop can have an adjacent color stop to its left, then the color stop which is adjacent to it to its left is the color stop from the set of color stops to its left which has an offset value which is closest to its offset value. If two or more color stops meet that criteria, then the color stop which is adjacent to it to its left is the color stop which has the highest index value from the set of color stops to its left which are tied for being closest to its offset value.

p) If a color stop can have an adjacent color stop to its right, then the color stop which is adjacent to it to its right is the color stop from the set of color stops to its right which has an offset value which is closest to its offset value. If two or more color stops meet that criteria, then the color stop which is adjacent to it to its right is the color stop which has the lowest index value from the set of color stops to its right which are tied for being closest to its offset value.

q) Where the value of o is in the range $[0, 1]$, its color value shall be determined by interpolating between the color stop, r, which is the color stop whose offset value is closest to o without being less than o and which can have an adjacent color stop to its left, and the color stop that is adjacent to r on r's left. The acceptable forms of interpolating between color values is set forth later in this section.

r) Where the value of o is outside the range $[0, 1]$, its color value depends on the value of wrap mode:

(1.1) — If wrap mode is `wrap_mode::none`, the color value of o shall be `rgba_color::transparent_-black()`.

(1.2) — If wrap mode is `wrap_mode::pad`, if o is negative then the color value of o shall be the same as-if the value of o was `0.0f`, otherwise the color value of o shall be the same as-if the value of o was `1.0f`.

(1.3) — If wrap mode is `wrap_mode::repeat`, then `1.0f` shall be added to or subtracted from o until o is in the range $[0, 1]$, at which point its color value is the color value for the modified value of o as determined by these rules. [*Example:* Given $o == 2.1$, after application of this rule $o == 0.1$ and the color value of o shall be the same value as-if the initial value of o was 0.1.

Given $o == -0.3$, after application of this rule $o == 0.7$ and the color value of o shall be the same as-if the initial value of o was 0.7. — *end example*]

(1.4) — If wrap mode is `wrap_mode::reflect`, o shall be set to the absolute value of o, then 2.0f shall be subtracted from o until o is in the range $[0, 2]$, then if o is in the range $(1, 2]$ then o shall be set to `1.0f - (o - 1.0f)`, at which point its color value is the color value for the modified value of o as determined by these rules. [*Example:* Given $o == 2.8$, after application of this rule $o == 0.8$ and the color value of o shall be the same value as-if the initial value of o was 0.8.

Given $o == 3.6$, after application of this rule $o == 0.4$ and the color value of o shall be the same value as-if the initial value of o was 0.4.

Given $o == -0.3$, after application of this rule $o == 0.3$ and the color value of o shall be the same as-if the initial value of o was 0.3.

Given $o == -5.8$, after application of this rule $o == 0.2$ and the color value of o shall be the same as-if the initial value of o was 0.2. — *end example*]

2 It is unspecified whether the interpolation between the color values of two adjacent color stops is performed linearly on each color channel or is performed by a linear color interpolation algorithm implemented in hardware (typically in a graphics processing unit).

3 Implementations shall interpolate between alpha channel values of adjacent color stops linearly except as provided in the following paragraph.

4 A conforming implementation may use the alpha channel interpolation results from a linear color interpolation algorithm implemented in hardware even if those results differ from the results required by the previous paragraph.

10.3 Enum class `wrap_mode` [io2d.wrapmode]

10.3.1 `wrap_mode` summary [io2d.wrapmode.summary]

1 The `wrap_mode` enum class describes how a point's visual data is determined if it is outside the bounds of the *source brush* (11.15.3.2) when sampling.

2 Depending on the source brush's `filter` value, the visual data of several points may be required to determine the appropriate visual data value for the point that is being sampled. In this case, each point shall be sampled according to the source brush's `wrap_mode` value with two exceptions:

a) If the point to be sampled is within the bounds of the source brush and the source brush's `wrap_mode` value is `wrap_mode::none`, then if the source brush's `filter` value requires that one or more points which are outside of the bounds of the source brush shall be sampled, each of those points shall be sampled as-if the source brush's `wrap_mode` value is `wrap_mode::pad` rather than `wrap_mode::none`.

b) If the point to be sampled is within the bounds of the source brush and the source brush's `wrap_mode` value is `wrap_mode::none`, ce Brush and the source brush's `wrap_mode` value is `wrap_mode::none`, then if the source brush's `filter` value requires that one or more points which are inside of the bounds of the source brush shall be sampled, each of those points shall be sampled such that the visual data that is returned shall be the equivalent of `rgba_color::transparent_black()`.

3 If a point to be sampled does not have a defined visual data element and the search for the nearest point with defined visual data produces two or more points with defined visual data that are equidistant from the point to be sampled, the returned visual data shall be an unspecified value which is the visual data of one of those equidistant points. Where possible, implementations should choose the among the equidistant points that have an x axisvalue and a y axisvalue that is nearest to `0.0f`.

4 See Table 4 for the meaning of each `wrap_mode` enumerator.

10.3.2 `wrap_mode` synopsis [io2d.wrapmode.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class wrap_mode {
    none,
    repeat,
    reflect,
    pad
  };
}
```

10.3.3 `wrap_mode` enumerators [io2d.wrapmode.enumerators]

Table 4 — `wrap_mode` enumerator meanings

Enumerator	Meaning
`none`	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned shall be the equivalent of `rgba_color::transparent_black()`.

Table 4 — `wrap_mode` enumerator meanings (continued)

Enumerator	Meaning
`repeat`	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned shall be the visual data that would have been returned if the source brush was infinitely large and repeated itself in a left-to-right-left-to-right and top-to-bottom-top-to-bottom fashion.
`reflect`	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned shall be the visual data that would have been returned if the source brush 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`	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned shall be the visual data that would have been returned for the nearest defined point that is in bounds.

10.4 Enum class `filter` [io2d.filter]

10.4.1 `filter` summary [io2d.filter.summary]

1 The `filter` enum class specifies the type of filter to use when sampling from a pixmap.

2 Three of the `filter` enumerators, `filter::fast`, `filter::good`, and `filter::best`, specify desired characteristics of the filter, leaving the choice of a specific filter to the implementation.

The other two, `filter::nearest` and `filter::bilinear`, each specify a particular filter that shall be used.

3 [*Note:* The only type of brush that has a pixmap as its underlying graphics data graphics resource is a brush with a brush type of `brush_type::surface`. — *end note*]

4 See Table 5 for the meaning of each `filter` enumerator.

10.4.2 `filter` synopsis [io2d.filter.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class filter {
    fast,
    good,
    best,
    nearest,
    bilinear
  };
}
```

10.4.3 `filter` enumerators [io2d.filter.enumerators]

Table 5 — `filter` enumerator meanings

Enumerator	Meaning
`fast`	The filter that corresponds to this value is implementation-defined. The implementation shall ensure that the time complexity of the chosen filter is not greater than the time complexity of the filter that corresponds to `filter::good`. [*Note:* By choosing this value, the user is hinting that performance is more important than quality. — *end note*]
`good`	The filter that corresponds to this value is implementation-defined. The implementation shall ensure that the time complexity of the chosen formula is not greater than the time complexity of the formula for `filter::best`. [*Note:* By choosing this value, the user is hinting that quality and performance are equally important. — *end note*]
`best`	The filter that corresponds to this value is implementation-defined. [*Note:* By choosing this value, the user is hinting that quality is more important than performance. — *end note*]
`nearest`	Nearest-neighbor interpolation filtering shall be used.
`bilinear`	Bilinear interpolation filtering shall be used.

10.5 Enum class `brush_type` [io2d.brushtype]

10.5.1 `brush_type` summary [io2d.brushtype.summary]

1 The `brush_type` enum class denotes the type of a `brush` object.

2 See Table 6 for the meaning of each `brush_type` enumerator.

10.5.2 `brush_type` synopsis [io2d.brushtype.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class brush_type {
    solid_color,
    surface,
    linear,
    radial
  };
}
```

10.5.3 `brush_type` enumerators [io2d.brushtype.enumerators]

Table 6 — `brush_type` enumerator meanings

Enumerator	Meaning
`solid_color`	The `brush` object is a solid color brush.
`surface`	The `brush` object is a surface brush.
`linear`	The `brush` object is a linear gradient brush.
`radial`	The `brush` object is a radial gradient brush.

10.6 Class `color_stop` [io2d.colorstop]

10.6.1 Overview [io2d.colorstop.intro]

1 The class `color_stop` describes a color stop that is used by gradient brushes.

2 It has an offset of type `float` and a color of type `rgba_color`.

10.6.2 `color_stop` synopsis [io2d.colorstop.synopsis]

```
namespace std::experimental::io2d::v1 {
  class color_stop {
  public:
        // 10.6.3, construct:
        constexpr color_stop() noexcept;
    constexpr color_stop(float o, const rgba_color& c) noexcept;

    // 10.6.4, modifiers:
    constexpr void offset(float o) noexcept;
        constexpr void color(const rgba_color& c) noexcept;

    // 10.6.5, observers:
        constexpr float offset() const noexcept;
        constexpr rgba_color color() const noexcept;
  };
}
```

10.6.3 `color_stop` constructors [io2d.colorstop.cons]

`constexpr color_stop() noexcept;`

1 *Effects:* Equivalent to: `color_stop(0.0f, rgba_color::transparent_black())`.

`constexpr color_stop(float o, const rgba_color& c) noexcept;`

2 *Requires:* `o >= 0.0f` and `o <= 1.0f`.

3 *Effects:* Constructs a `color_stop` object.

4 The offset is `o`. The color is `c`.

10.6.4 `color_stop` modifiers [io2d.colorstop.modifiers]

`constexpr void offset(float o) noexcept;`

1 *Requires:* `o >= 0.0f` and `o <= 1.0f`.

2 *Effects:* The offset is `o`.

`constexpr void color(const rgba_color& c) noexcept;`

3 *Effects:* The color is `c`.

10.6.5 `color_stop` observers [io2d.colorstop.observers]

`constexpr float offset() const noexcept;`

1 *Returns:* The offset.

`constexpr rgba_color color() const noexcept;`

2 *Returns:* The color.

10.7 Class brush [io2d.brush]

10.7.1 brush summary [io2d.brush.intro]

1 The class `brush` describes an opaque wrapper for graphics data.

2 A `brush` object is usable with any `surface` or `surface`-derived object.

3 A `brush` object's graphics data is immutable. It is observable only by the effect that it produces when the brush is used as a *source brush* or as a *mask brush* (11.15.3.2).

4 A `brush` object has a brush type of `brush_type`, which indicates which type of brush it is (Table 6).

5 As a result of technological limitations and considerations, a `brush` object's graphics data may have less precision than the data from which it was created.

10.7.2 brush synopsis [io2d.brush.synopsis]

```
namespace std::experimental::io2d::v1 {
  class brush {
  public:
    // 10.7.4, construct/copy/move/destroy:
    explicit brush(const rgba_color& c);
    template <class InputIterator>
    brush(const vector_2d& begin, const vector_2d& end,
      InputIterator first, InputIterator last);
    brush(const vector_2d& begin, const vector_2d& end,
      initializer_list<color_stop> il);
    template <class InputIterator>
    brush(const circle& start, const circle& end,
      InputIterator first, InputIterator last);
    brush(const circle& start, const circle& end,
      initializer_list<color_stop> il);
    explicit brush(image_surface&& img);

    // 10.7.5, observers:
    brush_type type() const noexcept;
  };
}
```

10.7.3 Sampling from a brush object [io2d.brush.sampling]

1 When sampling from a `brush` object `b`, the `brush_type` returned by calling `b.type()` shall determine how the results of sampling shall be determined:

1. If the result of `b.type()` is `brush_type::solid_color` then `b` is a *solid color brush*.
2. If the result of `b.type()` is `brush_type::surface` then `b` is a *surface brush*.
3. If the result of `b.type()` is `brush_type::linear` then `b` is a *linear gradient brush*.
4. If the result of `b.type()` is `brush_type::radial` then `b` is a *radial gradient brush*.

10.7.3.1 Sampling from a solid color brush [io2d.brush.sampling.color]

1 When `b` is a solid color brush, then when sampling from `b`, the visual data returned is always the visual data used to construct `b`, regardless of the point which is to be sampled and regardless of the return values of wrap mode, filter, and brush matrix or mask matrix.

10.7.3.2 Sampling from a linear gradient brush [io2d.brush.sampling.linear]

1 When `b` is a linear gradient brush, when sampling point `pt`, where `pt` is the return value of calling the `transform_point` member function of brush matrix or mask matrix using the requested point, from `b`, the

visual data returned are as specified by 10.2.2 and 10.2.4.

10.7.3.3 Sampling from a radial gradient brush [io2d.brush.sampling.radial]

1 When `b` is a radial gradient brush, when sampling point `pt`, where `pt` is the return value of calling the `transform_point` member function of brush matrix or mask matrix using the requested point, from `b`, the visual data are as specified by 10.2.3 and 10.2.4.

10.7.3.4 Sampling from a surface brush [io2d.brush.sampling.surface]

1 When `b` is a surface brush, when sampling point `pt`, where `pt` is the return value of calling the `transform_point` member function of brush matrix or mask matrix using the requested point, from `b`, the visual data returned are from the point `pt` in the graphics data of the brush, taking into account the values of wrap mode and filter.

10.7.4 `brush` constructors and assignment operators [io2d.brush.cons]

```
explicit brush(const rgba_color& c);
```

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

2 The brush's brush type shall be set to the value `brush_type::solid_color`.

3 The graphics data of the brush are created from the value of `c`. The visual data format of the graphics data are as-if it is that specified by `format::argb32`.

4 *Remarks:* Sampling from this produces the results specified in 10.7.3.1.

```
template <class InputIterator>
brush(const vector_2d& begin, const vector_2d& end,
  InputIterator first, InputIterator last);
```

5 *Effects:* Constructs a linear gradient `brush` object with a begin point of `begin`, an end point of `end`, and a sequential series of `color stop` values beginning at first and ending at last - 1.

6 The brush's brush type is `brush_type::linear`.

7 *Remarks:* Sampling from this brush produces the results specified in 10.7.3.2.

```
brush(const vector_2d& begin, const vector_2d& end,
  initializer_list<color_stop> il);
```

8 *Effects:* Constructs a linear gradient `brush` object with a begin point of `begin`, an end point of `end`, and the sequential series of `color stop` values in `il`.

9 The brush's brush type is `brush_type::linear`.

10 *Remarks:* Sampling from this brush produces the results specified in 10.7.3.2.

```
template <class InputIterator>
brush(const circle& start, const circle& end,
  InputIterator first, InputIterator last);
```

11 *Effects:* Constructs a radial gradient `brush` object with a start circle of `start`, an end circle of `end`, and a sequential series of `color stop` values beginning at first and ending at last - 1.

12 The brush's brush type is `brush_type::radial`.

13 *Remarks:* Sampling from this brush produces the results specified in 10.7.3.3.

```
brush(const circle& start, const circle& end,
  initializer_list<color_stop> il);
```

14 *Effects:* Constructs a radial gradient `brush` object with a start circle of `start`, an end circle of `end`, and the sequential series of `color stop` values in `il`.

15 The brush's brush type is `brush_type::radial`.

16 *Remarks:* Sampling from this brush produces the results specified in 10.7.3.3.

```
explicit brush(image_surface&& img);
```

18 *Effects:* Constructs an object of type `brush`.

19 The brush's brush type is `brush_type::surface`.

20 The graphics data of the brush is as-if it is the raster graphics data of `img`.

21 *Remarks:* Sampling from this brush produces the results specified in 10.7.3.4.

10.7.5 `brush` observers [io2d.brush.observers]

```
brush_type type() const noexcept;
```

1 *Returns:* The brush's brush type.

11 Surfaces [io2d.surfaces]

1 Surfaces are composed of visual data, stored in a graphics data graphics resource. [*Note:* All well-defined `surface`-derived types are currently raster graphics data graphics resources with defined bounds. To allow for easier additions of future surface-derived types which are not composed of raster graphics data or do not have fixed bounds, such as a vector graphics-based surface, the less constrained term graphics data graphics resource is used. — *end note*]

2 The surface's visual data is manipulated by rendering and composing operations (11.15.3).

3 Surfaces are stateful objects.

4 The various `surface`-derived classes each provide specific, unique functionality that enables a broad variety of 2D graphics operations to be accomplished efficiently.

11.1 Enum class `antialias` [io2d.antialias]

11.1.1 `antialias` summary [io2d.antialias.summary]

1 The antialias enum class specifies the type of anti-aliasing that the rendering system uses for rendering and composing paths. See Table 7 for the meaning of each `antialias` enumerator.

11.1.2 `antialias` synopsis [io2d.antialias.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class antialias {
    none,
    fast,
    good,
    best
  };
}
```

11.1.3 `antialias` enumerators [io2d.antialias.enumerators]

Table 7 — `antialias` enumerator meanings

Enumerator	Meaning
	No anti-aliasing is performed.
`fast`	Some form of anti-aliasing shall be used when this option is selected, but the form used is implementation-defined. [*Note:* By specifying this value, the user is hinting that faster anti-aliasing is preferable to better anti-aliasing. — *end note*]
`good`	Some form of anti-aliasing shall be used when this option is selected, but the form used is implementation-defined. [*Note:* By specifying this value, the user is hinting that sacrificing some performance to obtain better anti-aliasing is acceptable but that performance is still a concern. — *end note*]

Table 7 — `antialias` enumerator meanings (continued)

Enumerator	Meaning
`best`	Some form of anti-aliasing shall be used when this option is selected, but the form used is implementation-defined. [*Note:* By specifying this value, the user is hinting that anti-aliasing is more important than performance. — *end note*]

11.2 Enum class `fill_rule` [io2d.fillrule]

11.2.1 `fill_rule` summary [io2d.fillrule.summary]

1 The `fill_rule` enum class determines how the filling operation (11.15.6) is performed on a path group.

2 For each point, draw a ray from that point to infinity which does not pass through the start point or end point of any non-degenerate path segment in the path group, is not tangent to any non-degenerate path segment in the path group, and is not coincident with any non-degenerate path segment in the path group.

3 See Table 8 for the meaning of each `fill_rule` enumerator.

11.2.2 `fill_rule` synopsis [io2d.fillrule.synopsis]

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

11.2.3 `fill_rule` enumerators [io2d.fillrule.enumerators]

Table 8 — `fill_rule` enumerator meanings

Enumerator	Meaning
`winding`	If the *fill rule* (11.11.1) is `fill_rule::winding`, then using the ray described above and beginning with a count of zero, add one to the count each time a non-degenerate path segment crosses the ray going left-to-right from its begin point to its end point, and subtract one each time a non-degenerate path segment crosses the ray going from right-to-left from its begin point to its end point. If the resulting count is zero after all non-degenerate path segments that cross the ray have been evaluated, the point shall not be filled; otherwise the point shall be filled.
`even_odd`	If the fill rule is `fill_rule::even_odd`, then using the ray described above and beginning with a count of zero, add one to the count each time a non-degenerate path segment crosses the ray. If the resulting count is an odd number after all non-degenerate path segments that cross the ray have been evaluated, the point shall be filled; otherwise the point shall not be filled. [*Note:* Mathematically, zero is an even number, not an odd number. — *end note*]

11.3 Enum class `line_cap` [io2d.linecap]

11.3.1 `line_cap` summary [io2d.linecap.summary]

1 The `line_cap` enum class specifies how the ends of lines should be rendered when a `path_group` object is stroked. See Table 9 for the meaning of each `line_cap` enumerator.

11.3.2 `line_cap` synopsis [io2d.linecap.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class line_cap {
    none,
    round,
    square
  };
}
```

11.3.3 `line_cap` enumerators [io2d.linecap.enumerators]

Table 9 — `line_cap` enumerator meanings

Enumerator	Meaning
`none`	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.

11.4 Enum class `line_join` [io2d.linejoin]

11.4.1 `line_join` summary [io2d.linejoin.summary]

1 The `line_join` enum class specifies how the junction of two line segments should be rendered when a `path_group` is stroked. See Table 10 for the meaning of each enumerator.

11.4.2 `line_join` synopsis [io2d.linejoin.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class line_join {
    miter,
    round,
    bevel
  };
}
```

11.4.3 `line_join` enumerators [io2d.linejoin.enumerators]

Table 10 — `line_join` enumerator meanings

Enumerator	Meaning
`miter`	Joins will be mitered or beveled, depending on the miter limit (see: 11.13.1).
`round`	Joins will be rounded, with the center of the circle being the join point.

Table 10 — `line_join` enumerator meanings (continued)

Enumerator	Meaning
`bevel`	Joins will be beveled, with the join cut off at half the line width from the join point. Implementations may vary the cut off distance by an amount that is less than one pixel at each join for aesthetic or technical reasons.

11.5 Enum class `compositing_op` [io2d.compositingop]

11.5.1 `compositing_op` Summary [io2d.compositingop.summary]

1 The `compositing_op` enum class specifies composition algorithms. See Table 11, Table 12 and Table 13 for the meaning of each `compositing_op` enumerator.

11.5.2 `compositing_op` Synopsis [io2d.compositingop.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class compositing_op {
    // basic
    over,
    clear,
    source,
    in,
    out,
    atop,
    dest,
    dest_over,
    dest_in,
    dest_out,
    dest_atop,
    xor_op,
    add,
    saturate,
    // blend
    multiply,
    screen,
    overlay,
    darken,
    lighten,
    color_dodge,
    color_burn,
    hard_light,
    soft_light,
    difference,
    exclusion,
    // hsl
    hsl_hue,
    hsl_saturation,
    hsl_color,
    hsl_luminosity
  };
}
```

11.5.3 `compositing_op` Enumerators [io2d.compositingop.enumerators]

1 The tables below specifies the mathematical formula for each enumerator's composition algorithm. The formulas differentiate between three color channels (red, green, and blue) and an alpha channel (transparency). For all channels, valid channel values are in the range $[0.0, 1.0]$.

2 Where a visual data format for a visual data element has no alpha channel, the visual data format shall be treated as though it had an alpha channel with a value of 1.0 for purposes of evaluating the formulas.

3 Where a visual data format for a visual data element has no color channels, the visual data format shall be treated as though it had a value of 0.0 for all color channels for purposes of evaluating the formulas.

4 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

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

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

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

8 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).

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

10 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*]

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

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

13 The basic enumerators specify a value for *bound*. This value may be 'Yes', 'No', or 'N/A'.

14 If the bound value is 'Yes', 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.

15 If the bound value is 'No', 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.

16 If the bound value is 'N/A', the operation would have the same effect regardless of whether it was treated as 'Yes' or 'No' such that those bound values are not applicable to the operation. A '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. 'Yes' and 'No' treatment each produce the same result in areas where the source does not have a value.

17 If a clip is set and the bound value is 'Yes' or 'N/A', then only those areas of the surface that the are within the clip will be affected by the compositing operation.

18 If a clip is set and the bound value is 'No', 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.

Table 11 — `compositing_op` basic enumerator meanings

Enumerator	Bound	Color	Alpha
`clear`	Yes	$Rc = 0$	$Ra = 0$
`source`	Yes	$Rc = Sc$	$Ra = Sa$
`over`	N/A	$Rc = \frac{(Sca + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da \times (1 - Sa)$
`in`	No	$Rc = Sc$	$Ra = Sa \times Da$
`out`	No	$Rc = Sc$	$Ra = Sa \times (1 - Da)$
`atop`	N/A	$Rc = Sca + Dc \times (1 - Sa)$	$Ra = Da$
`dest`	N/A	$Rc = Dc$	$Ra = Da$
`dest_over`	N/A	$Rc = \frac{(Sca \times (1 - Da) + Dca)}{Ra}$	$Ra = (1 - Da) \times Sa + Da$
`dest_in`	No	$Rc = Dc$	$Ra = Sa \times Da$
`dest_out`	N/A	$Rc = Dc$	$Ra = (1 - Sa) \times Da$
`dest_atop`	No	$Rc = Sc \times (1 - Da) + Dca$	$Ra = Sa$
`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	$Rc = \frac{(min(Sa, 1 - Da) \times Sc + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$

19 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.

20 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.

21 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), 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*]

22 For the enumerators in Table 12 and Table 13 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*]

23 All of the blend enumerators and hsl enumerators have a bound value of 'N/A'.

Table 12 — `compositing_op` blend enumerator meanings

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

`overlay`	$if(Dc \leq 0.5f)$ {
	$f(Sc, Dc) = 2 \times Sc \times Dc$
	}
	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 {
	$f(Sc, Dc) = 0$
	}

Table 12 — `compositing_op` blend enumerator meanings (continued)

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

24 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).

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

26 $min(x,\ y,\ z)\ =\ min(x,\ min(y,\ z))$

27 $max(x,\ y,\ z)\ =\ max(x,\ max(y,\ z))$

28 $sat(C) = max(Cr,\ Cg,\ Cb) - min(Cr,\ Cg,\ Cb)$

29 $lum(C) = Cr \times 0.3 + Cg \times 0.59 + Cb \times 0.11$

30 $clip\_color(C) =\ \{$
$L = lum(C)$

$N = min(Cr, Cg, Cb)$
$X = max(Cr, Cg, Cb)$
$if\ (N < 0.0)\ \{$
$Cr = L + \frac{((Cr - L) \times L)}{(L - N)}$
$Cg = L + \frac{((Cg - L) \times L)}{(L - N)}$
$Cb = L + \frac{((Cb - L) \times L)}{(L - N)}$
$\}$
$if\ (X > 1.0)\ \{$
$Cr = L + \frac{((Cr - L) \times (1 - L))}{(X - L)}$
$Cg = L + \frac{((Cg - L) \times (1 - L))}{(X - L)}$
$Cb = L + \frac{((Cb - L) \times (1 - L))}{(X - L)}$
$\}$
$return\ C$
$\}$

31 $set\_lum(C, L) = \ \{$
$D = L - lum(C)$
$Cr = Cr + D$
$Cg = Cg + D$
$Cb = Cb + D$
$return\ clip\_color(C)$
$\}$

32 $set\_sat(C, S) = \ \{$
$R = C$
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 13 — `compositing_op` 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`	$f(Sc, Dc) = set\_lum(Dc,\ lum(Sc))$

11.6 Enum class `format` [io2d.format]

11.6.1 `format` summary [io2d.format.summary]

1 The `format` enum class indicates a visual data format. See Table 14 for the meaning of each `format` enumerator.

2 Unless otherwise specified, a visual data format shall be an unsigned integral value of the specified bit size in native-endian format.

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, with the maximum value representing the maximum contribution from that channel. [*Example:* Given a 5-bit channel representing the color , 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.

A — *end example*]

11.6.2 `format` synopsis [io2d.format.synopsis]

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

11.6.3 `format` enumerators [io2d.format.enumerators]

Table 14 — `format` enumerator meanings

Enumerator	Meaning
`invalid`	A previously specified `format` is unsupported by the implementation.
`argb32`	A 32-bit RGB color model 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. This is a premultiplied format.
`rgb24`	A 32-bit RGB color model pixel format. The upper 8 bits are unused, followed by an 8-bit red color channel, then an 8-bit green color channel, and finally an 8-bit blue color channel.

Table 14 — `format` enumerator meanings (continued)

Enumerator	Meaning
`a8`	An 8-bit transparency data pixel format. All 8 bits are an alpha channel.
`rgb16_565`	A 16-bit RGB color model pixel format. The upper 5 bits are a red color channel, followed by a 6-bit green color channel, and finally a 5-bit blue color channel.
`rgb30`	A 32-bit RGB color model pixel format. The upper 2 bits are unused, followed by a 10-bit red color channel, a 10-bit green color channel, and finally a 10-bit blue color channel. The value in each channel is an unsigned normalized integer.

11.7 Enum class `scaling` [io2d.scaling]

11.7.1 `scaling` summary [io2d.scaling.summary]

1 The scaling enum class specifies the type of scaling a `display_surface` will use when the size of its *display buffer* (11.17.1) differs from the size of its *back buffer* (11.17.1).

2 See Table 15 for the meaning of each `scaling` enumerator.

11.7.2 `scaling` synopsis [io2d.scaling.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class scaling {
    letterbox,
    uniform,
    fill_uniform,
    fill_exact,
    none
  };
}
```

11.7.3 `scaling` enumerators [io2d.scaling.enumerators]

1 [*Note:* In the following table, examples will be given to help explain the meaning of each enumerator. The examples will all use a `display_surface` called `ds`.

The back buffer (11.17.1) of `ds` is 640x480 (i.e. it has a width of 640 pixels and a height of 480 pixels), giving it an aspect ratio of $1.\bar{3}$.

The display buffer (11.17.1) of `ds` is 1280x720, giving it an aspect ratio of $1.\bar{7}$.

When a rectangle is defined in an example, the coordinate $(x1, y1)$ denotes the top left corner of the rectangle, inclusive, and the coordinate $(x2, y2)$ denotes the bottom right corner of the rectangle, exclusive. As such, a rectangle with $(x1, y1) = (10, 10)$, $(x2, y2) = (20, 20)$ is 10 pixels wide and 10 pixels tall and includes the pixel $(x, y) = (19, 19)$ but does not include the pixels $(x, y) = (20, 19)$ or $(x, y) = (19, 20)$. *— end note*]

Table 15 — `scaling` enumerator meanings

Enumerator	Meaning
`letterbox`	Fill the display buffer with the letterbox brush (11.17.4) of the `display_surface`. Uniformly scale the back buffer so that one dimension of it is the same length as the same dimension of the display buffer and the second dimension of it is not longer than the second dimension of the display buffer and transfer the scaled back buffer to the display buffer using sampling such that it is centered in the display buffer. [*Example:* The display buffer of `ds` will be filled with the `brush` object returned by `ds.letterbox_brush();`. The back buffer of `ds` will be scaled so that it is 960x720, thereby retaining its original aspect ratio. The scaled back buffer will be transfered to the display buffer using sampling such that it is in the rectangle $(x1, y1) = (\frac{1280}{2} - \frac{960}{2}, 0) = (160, 0)$, $(x2, y2) = (960 + (\frac{1280}{2} - \frac{960}{2}), 720) = (1120, 720)$. This fulfills all of the conditions. At least one dimension of the scaled back buffer is the same length as the same dimension of the display buffer (both have a height of 720 pixels). The second dimension of the scaled back buffer is not longer than the second dimension of the display buffer (the back buffer's scaled width is 960 pixels, which is not longer than the display buffer's width of 1280 pixels. Lastly, the scaled back buffer is centered in the display buffer (on the x axis there are 160 pixels between each vertical side of the scaled back buffer and the nearest vertical edge of the display buffer and on the y axis there are 0 pixels between each horizontal side of the scaled back buffer and the nearest horizontal edge of the display buffer). — *end example*]

Table 15 — `scaling` enumerator meanings (continued)

Enumerator	Meaning
`uniform`	Uniformly scale the back buffer so that one dimension of it is the same length as the same dimension of the display buffer and the second dimension of it is not longer than the second dimension of the display buffer and transfer the scaled back buffer to the display buffer using sampling such that it is centered in the display buffer. [*Example:* The back buffer of `ds` will be scaled so that it is 960x720, thereby retaining its original aspect ratio. The scaled back buffer will be transfered to the display buffer using sampling such that it is in the rectangle $(x1, y1) = (\frac{1280}{2} - \frac{960}{2}, 0) = (160, 0)$, $(x2, y2) = (960 + (\frac{1280}{2} - \frac{960}{2}), 720) = (1120, 720)$. This fulfills all of the conditions. At least one dimension of the scaled back buffer is the same length as the same dimension of the display buffer (both have a height of 720 pixels). The second dimension of the scaled back buffer is not longer than the second dimension of the display buffer (the back buffer's scaled width is 960 pixels, which is not longer than the display buffer's width of 1280 pixels. Lastly, the scaled back buffer is centered in the display buffer (on the x axis there are 160 pixels between each vertical side of the scaled back buffer and the nearest vertical edge of the display buffer and on the y axis there are 0 pixels between each horizontal side of the scaled back buffer and the nearest horizontal edge of the display buffer). — *end example*] [*Note:* The difference between `uniform` and `letterbox` is that `uniform` does not modify the contents of the display buffer that fall outside of the rectangle into which the scaled back buffer is drawn while `letterbox` fills those areas with the `display_surface` object's letterbox brush (see: 11.17.4). — *end note*]

Table 15 — `scaling` enumerator meanings (continued)

Enumerator	Meaning
`fill_uniform`	Uniformly scale the back buffer so that one dimension of it is the same length as the same dimension of the display buffer and the second dimension of it is not shorter than the second dimension of the display buffer and transfer the scaled back buffer to the display buffer using sampling such that it is centered in the display buffer. [*Example:* The back buffer of `ds` will be drawn in the rectangle $(x1, y1) = (0, -120)$, $(x2, y2) = (1280, 840)$. This fulfills all of the conditions. At least one dimension of the scaled back buffer is the same length as the same dimension of the display buffer (both have a width of 1280 pixels). The second dimension of the scaled back buffer is not shorter than the second dimension of the display buffer (the back buffer's scaled height is 840 pixels, which is not shorter than the display buffer's height of 720 pixels). Lastly, the scaled back buffer is centered in the display buffer (on the x axis there are 0 pixels between each vertical side of the rectangle and the nearest vertical edge of the display buffer and on the y axis there are 120 pixels between each horizontal side of the rectangle and the nearest horizontal edge of the display buffer). — *end example*]
`fill_exact`	Scale the back buffer so that each dimension of it is the same length as the same dimension of the display buffer and transfer the scaled back buffer to the display buffer using sampling such that its origin is at the origin of the display buffer. [*Example:* The back buffer will be drawn in the rectangle $(x1, y1) = (0, 0)$, $(x2, y2) = (1280, 720)$. This fulfills all of the conditions. Each dimension of the scaled back buffer is the same length as the same dimension of the display buffer (both have a width of 1280 pixels and a height of 720 pixels) and the origin of the scaled back buffer is at the origin of the display buffer. — *end example*]
`none`	Do not perform any scaling. Transfer the back buffer to the display buffer using sampling such that its origin is at the origin of the display buffer. [*Example:* The back buffer of `ds` will be drawn in the rectangle $(x1, y1) = (0, 0)$, $(x2, y2) = (640, 480)$ such that no scaling occurs and the origin of the back buffer is at the origin of the display buffer. — *end example*]

11.8 Enum class `refresh_rate` [io2d.refreshrate]

11.8.1 `refresh_rate` summary [io2d.refreshrate.summary]

1 The `refresh_rate` enum class describes when the *draw callback* (Table 22) of a `display_surface` object shall be called. See Table 16 for the meaning of each enumerator.

11.8.2 `refresh_rate` synopsis [io2d.refreshrate.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class refresh_rate {
    as_needed,
    as_fast_as_possible,
    fixed
  };
}
```

11.8.3 `refresh_rate` enumerators [io2d.refreshrate.enumerators]

Table 16 — `refresh_rate` value meanings

Enumerator	Meaning
`as_needed`	The draw callback shall be called when the implementation needs to do so. [*Note:* The intention of this enumerator is that implementations will call the draw callback as little as possible in order to minimize power usage. Users can call `display_surface::redraw_required` to make the implementation run the draw callback whenever the user requires. — *end note*]
`as_fast_as_possible`	The draw callback shall be called as frequently as possible, subject to any limits of the execution environment and the underlying rendering and presentation technologies.
`fixed`	The draw callback shall be called as frequently as needed to maintain the *desired frame rate* (Table 22) as closely as possible. If more time has passed between two successive calls to the draw callback than is required, it shall be called *excess time* and it shall count towards the *required time*, which is the time that is required to pass after a call to the draw callback before the next successive call to the draw callback shall be made. If the excess time is greater than the required time, implementations shall call the draw callback and then repeatedly subtract the required time from the excess time until the excess time is less than the required time. If the implementation needs to call the draw callback for some other reason, it shall use that call as the new starting point for maintaining the desired frame rate. [*Example:* Given a desired frame rate of `20.0f`, then as per the above, the implementation would call the draw callback at 50 millisecond intervals or as close thereto as possible. If for some reason the excess time is 51 milliseconds, the implementation would call the draw callback, subtract 50 milliseconds from the excess time, and then would wait 49 milliseconds before calling the draw callback again. If only 15 milliseconds have passed since the draw callback was last called and the implementation needs to call the draw callback again, then the implementation shall call the draw callback immediately and proceed to wait 50 milliseconds before calling the draw callback again. — *end example*]

11.9 Enum class `image_file_format` [io2d.imagefileformat]

11.9.1 `image_file_format` summary [io2d.imagefileformat.summary]

1 The `image_file_format` enum class specifies the data format that an `image_surface` object is constructed from or saved to. This allows data in a format that is required to be supported to be read or written regardless of its extension.

2 It also has a value that allows implementations to support additional file formats if it recognizes them.

11.9.2 `image_file_format` synopsis [io2d.imagefileformat.synopsis]

```
namespace std::experimental::io2d::v1 {
  enum class image_file_format {
    unknown,
    png,
    jpg,
    tiff
  };
}
```

11.9.3 `image_file_format` enumerators [io2d.imagefileformat.enumerators]

Table 17 — `imagefileformat` enumerator meanings

Enumerator	Meaning
`unknown`	The format is unknown because it is not an image file format that is required to be supported. It may be known and supported by the implementation.
`png`	The PNG format.
`jpg`	The JPEG format.
`tiff`	The TIFF format.

11.10 Class `render_props` [io2d.renderprops]

11.10.1 `render_props` summary [io2d.renderprops.summary]

1 The `render_props` class provides general state information that is applicable to all rendering and composing operations (11.15.3).

2 It has an *antialias* of type `antialias`, a *surface matrix* of type `matrix_2d`, and a *compositing operator* of type `compositing_op`.

11.10.2 `render_props` synopsis [io2d.renderprops.synopsis]

```
namespace std::experimental::io2d::v1 {
  class render_props {
  public:
    // 11.10.3, constructors:
    constexpr render_props() noexcept;
    constexpr explicit render_props(antialias a, const matrix_2d& m =
      matrix_2d{ }, compositing_op co = compositing_op::over) noexcept;

    // 11.10.4, modifiers:
    constexpr void antialiasing(antialias a) noexcept;
    constexpr void compositing(compositing_op co) noexcept;
    constexpr void surface_matrix(const matrix_2d& m) noexcept;
```

```
    // 11.10.5, observers:
    constexpr antialias antialiasing() const noexcept;
    constexpr compositing_op compositing() const noexcept;
    constexpr matrix_2d surface_matrix() const noexcept;
  };
}
```

11.10.3 render_props constructors [io2d.renderprops.cons]

`constexpr render_props() noexcept;`

1 *Effects:* Equivalent to: `render_props(antialias::good)`.

```
constexpr explicit render_props(antialias a, const matrix_2d& m,
  compositing_op co) noexcept;
```

2 *Requires:* `m.is_invertible() == true`.

3 *Effects:* The antialias is `a`. The surface matrix is `m`. The compositing operator is `co`.

11.10.4 render_props modifiers [io2d.renderprops.modifiers]

`constexpr void antialiasing(antialias a) noexcept;`

1 *Effects:* The antialias is `a`.

`constexpr void compositing(compositing_op co) noexcept;`

2 *Effects:* The compositing operator is `co`.

`constexpr void surface_matrix(const matrix_2d& m) noexcept;`

3 *Requires:* `m.is_invertible() == true`.

4 *Effects:* The surface matrix is `m`.

11.10.5 render_props observers [io2d.renderprops.observers]

`constexpr antialias antialiasing() const noexcept;`

1 *Returns:* The antialias.

`constexpr compositing_op compositing() const noexcept;`

2 *Returns:* The compositing operator.

`constexpr matrix_2d surface_matrix() const noexcept;`

3 *Returns:* The surface matrix.

11.11 Class brush_props [io2d.brushprops]

11.11.1 brush_props summary [io2d.brushprops.summary]

1 The `brush_props` class provides general state information that is applicable to all rendering and composing operations (11.15.3).

2 It has a *wrap mode* of type `wrap_mode`, a *filter* of type `filter`, a *fill rule* of type `fill_rule`, and a *brush matrix* of type `matrix_2d`.

11.11.2 brush_props synopsis [io2d.brushprops.synopsis]

```
namespace std::experimental::io2d::v1 {
  class brush_props {
  public:
```

```
    // 11.11.3, constructors:
    constexpr brush_props(
      experimental::io2d::filter fi = experimental::io2d::filter::good,
      experimental::io2d::wrap_mode w = experimental::io2d::wrap_mode::none,
      experimental::io2d::fill_rule fr = experimental::io2d::fill_rule::winding,
      matrix_2d m = matrix_2d{}) noexcept;

    // 11.11.4, modifiers:
    constexpr void filter(experimental::io2d::filter fi) noexcept;
    constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
    constexpr void fill_rule(experimental::io2d::fill_rule fr) noexcept;
    constexpr void brush_matrix(const matrix_2d& m) noexcept;

    // 11.11.5, observers:
    constexpr experimental::io2d::filter filter() const noexcept;
    constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
    constexpr experimental::io2d::fill_rule fill_rule() const noexcept;
    constexpr matrix_2d brush_matrix() const noexcept;
  };
}
```

11.11.3 brush_props constructors [io2d.brushprops.cons]

```
constexpr brush_props(
  experimental::io2d::wrap_mode w = experimental::io2d::wrap_mode::none,
  experimental::io2d::filter fi = experimental::io2d::filter::good,
  experimental::io2d::fill_rule fr = experimental::io2d::fill_rule::winding,
  matrix_2d m = matrix_2d{}) noexcept
```

Requires: `m.is_invertible() == true`.

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

2 The wrap mode is `w`. The filter is `fi`. The fill rule is `fr`. The brush matrix is `m`.

11.11.4 brush_props modifiers [io2d.brushprops.modifiers]

```
constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
```

1 *Effects:* The wrap mode is `w`.

```
constexpr void filter(experimental::io2d::filter fi) noexcept;
```

2 *Effects:* The filter is `fi`.

```
constexpr void fill_rule(experimental::io2d::fill_rule fr) noexcept;
```

3 *Effects:* The fill rule is `fr`.

```
constexpr void brush_matrix(const matrix_2d& m) noexcept;
```

4 *Requires:* `m.is_invertible() == true`.

5 *Effects:* The brush matrix is `m`.

11.11.5 brush_props observers [io2d.brushprops.observers]

```
constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
```

1 *Returns:* The wrap mode.

```
constexpr experimental::io2d::filter filter() const noexcept;
```

2 *Returns:* The filter.

```
constexpr experimental::io2d::fill_rule fill_rule() const noexcept;
```

3 *Returns:* The fill rule.

```
constexpr matrix_2d brush_matrix() const noexcept;
```

4 *Returns:* The brush matrix.

11.12 Class clip_props [io2d.clipprops]

11.12.1 clip_props summary [io2d.clipprops.summary]

1 The `clip_props` class provides general state information that is applicable to all rendering and composing operations (11.15.3).

2 It has a *clip area* of type `path_group` and a *fill rule* of type `fill_rule`.

11.12.2 clip_props synopsis [io2d.clipprops.synopsis]

```
namespace std::experimental::io2d::v1 {
  class clip_props {
  public:
    // 11.12.3, constructors:
    clip_props() noexcept;
    template <class Allocator>
    explicit clip_props(const path_builder<Allocator>& pb,
      experimental::io2d::fill_rule fr =
      experimental::io2d::fill_rule::winding);
    explicit clip_props(const path_group& pg, experimental::io2d::fill_rule fr =
```

```
      experimental::io2d::fill_rule::winding) noexcept;

    // 11.12.4, modifiers:
    template <class Allocator>
    void clip(const path_builder<Allocator>& pb);
    void clip(const path_group& pg) noexcept;
    void fill_rule(experimental::io2d::fill_rule fr) noexcept;

    // 11.12.5, observers:
    path_group clip() const noexcept;
    experimental::io2d::fill_rule fill_rule() const noexcept;
  };
}
```

11.12.3 clip_props constructors [io2d.clipprops.cons]

```
clip_props() noexcept;
```

1 *Effects:* Equivalent to: `clip_props(path_builder<>{ })`.

```
template <class Allocator>
explicit clip_props(const path_builder<Allocator>& pb,
  experimental::io2d::fill_rule fr);
explicit clip_props(const path_group& pg, experimental::io2d::fill_rule fr)
  noexcept;
```

2 *Effects:* Constructs an object of type `clip_props`.

3 The clip area is:

(3.1) — `path_group{pb}`; or

(3.2) — `pg`.

4 The fill rule is `fr`.

11.12.4 clip_props modifiers [io2d.clipprops.modifiers]

```
template <class Allocator>
void clip(const path_builder<Allocator>& pb, experimental::io2d::fill_rule fr);
void clip(const path_group& pg, experimental::io2d::fill_rule fr) noexcept;
```

1 *Effects:* The clip area is:

(1.1) — `path_group{pb}`; or

(1.2) — `pg`.

```
void fill_rule(experimental::io2d::fill_rule fr) noexcept;
```

2 *Effects:* The fill rule is `fr`.

11.12.5 clip_props observers [io2d.clipprops.observers]

```
path_group clip() const noexcept;
```

1 *Returns:* The clip area.

```
experimental::io2d::fill_rule fill_rule() const noexcept;
```

2 *Returns:* The fill rule.

11.13 Class stroke_props [io2d.strokeprops]

11.13.1 stroke_props summary [io2d.strokeprops.summary]

1 The `stroke_props` class provides state information that is applicable to the stroking operation (see: 11.15.3 and 11.15.7).

2 It has a *line width* of type `float`, a *line cap* of type `line_cap`, a *line join* of type `line_join`, and a *miter limit* of type `float`.

11.13.2 stroke_props synopsis [io2d.strokeprops.synopsis]

```
namespace std::experimental::io2d::v1 {
  class stroke_props {
  public:
    // 11.13.3, constructors:
    constexpr stroke_props() noexcept;
    constexpr explicit stroke_props(float w,
      experimental::io2d::line_cap lc = experimental::io2d::line_cap::none,
      experimental::io2d::line_join lj = experimental::io2d::line_join::miter,
      float ml = 10.0f) noexcept

    // 11.13.4, modifiers:
    constexpr void line_width(float w) noexcept;
    constexpr void line_cap(experimental::io2d::line_cap lc) noexcept;
    constexpr void line_join(experimental::io2d::line_join lj) noexcept;
    constexpr void miter_limit(float ml) noexcept;

    // 11.13.5, observers:
    constexpr float line_width() const noexcept;
    constexpr experimental::io2d::line_cap line_cap() const noexcept;
    constexpr experimental::io2d::line_join line_join() const noexcept;
    constexpr float miter_limit() const noexcept;
    constexpr float max_miter_limit() const noexcept;
  };
}
```

11.13.3 stroke_props constructors [io2d.strokeprops.cons]

```
constexpr stroke_props() noexcept;
```

1 *Effects:* Equivalent to: `stroke_props(2.0f)`.

```
constexpr explicit stroke_props(float w,
  experimental::io2d::line_cap lc = experimental::io2d::line_cap::none,
  experimental::io2d::line_join lj = experimental::io2d::line_join::miter,
  float ml = 10.0f) noexcept
```

2 *Requires:* `w > 0.0f`. `ml >= 10.0f`. `ml <= max_miter_limit()`.

3 *Effects:* The line width is `w`. The line cap is `lc`. The line join is `lj`. The miter limit is `ml`.

11.13.4 stroke_props modifiers [io2d.strokeprops.modifiers]

```
constexpr void line_width(float w) noexcept;
```

1 *Requires:* `w >= 0.0f`.

2 *Effects:* The line width is `w`.

```
constexpr void line_cap(experimental::io2d::line_cap lc) noexcept;
```

3 *Effects:* The line cap is `lc`.

```
constexpr void line_join(experimental::io2d::line_join lj) noexcept;
```

4 *Effects:* The line join is `lj`.

```
constexpr void miter_limit(float ml) noexcept;
```

5 *Requires:* `ml >= 1.0f` and `ml <= max_miter_limit`.

6 *Effects:* The miter limit is `ml`.

11.13.5 `stroke_props` observers [io2d.strokeprops.observers]

```
constexpr float line_width() const noexcept;
```

1 *Returns:* The line width.

```
constexpr experimental::io2d::line_cap line_cap() const noexcept;
```

2 *Returns:* The line cap.

```
constexpr experimental::io2d::line_join line_join() const noexcept;
```

3 *Returns:* The line join.

```
constexpr float miter_limit() const noexcept;
```

4 *Returns:* The miter limit.

```
constexpr float max_miter_limit() const noexcept;
```

5 *Requires:* This value shall be finite and greater than `10.0f`.

6 *Returns:* The implementation-defined maximum value of miter limit.

11.14 Class `mask_props` [io2d.maskprops]

11.14.1 `mask_props` summary [io2d.maskprops.summary]

1 The `mask_props` class provides state information that is applicable to the mask rendering and composing operation (11.15.3).

2 It has a *wrap mode* of type `wrap_mode`, a *filter* of type `filter`, and a *mask matrix* of type `matrix_2d`.

11.14.2 `mask_props` synopsis [io2d.maskprops.synopsis]

```
namespace std::experimental::io2d::v1 {
  class mask_props {
  public:
    // 11.14.3, constructors:
    constexpr mask_props(
      experimental::io2d::wrap_mode w = experimental::io2d::wrap_mode::repeat,
      experimental::io2d::filter fi = experimental::io2d::filter::good,
      matrix_2d m = matrix_2d{}) noexcept;

    // 11.14.4, modifiers:
    constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
    constexpr void filter(experimental::io2d::filter fi) noexcept;
    constexpr void mask_matrix(const matrix_2d& m) noexcept;

    // 11.14.5, observers:
    constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
    constexpr experimental::io2d::filter filter() const noexcept;
```

```
      constexpr matrix_2d mask_matrix() const noexcept;
    };
  }
```

11.14.3 `mask_props` constructors [io2d.maskprops.cons]

```
constexpr mask_props (experimental::io2d::wrap_mode w,
  experimental::io2d::filter fi, matrix_2d m) noexcept;
```

Requires: `m.is_invertible() == true`.

1 *Effects:* The wrap mode is `w`. The filter is `fi`. The mask matrix is `m`.

11.14.4 `mask_props` modifiers [io2d.maskprops.modifiers]

```
constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
```

1 *Effects:* The wrap mode is `w`.

```
constexpr void filter(experimental::io2d::filter fi) noexcept;
```

2 *Effects:* The filter is `fi`.

```
constexpr void mask_matrix(const matrix_2d& m) noexcept;
```

3 *Requires:* `m.is_invertible() == true`.

4 *Effects:* The mask matrix is `m`.

11.14.5 `mask_props` observers [io2d.maskprops.observers]

```
constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
```

1 *Returns:* The wrap mode.

```
constexpr experimental::io2d::filter filter() const noexcept;
```

2 *Returns:* The filter.

```
constexpr matrix_2d mask_matrix() const noexcept;
```

3 *Returns:* The mask matrix.

11.15 Class `surface` [io2d.surface]

11.15.1 `surface` description [io2d.surface.intro]

1 The `surface` class provides an interface for managing a graphics data graphics resource.

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

3 The `surface` class provides two ways to modify its graphics resource:

(3.1) — Rendering and composing operations.

(3.2) — Mapping.

4 [*Note:* While a `surface` object manages a graphics data graphics resource, the `surface` class does not provide well-defined semantics for the graphics resource. The `surface` class is intended to serve only as a base class and as such is not directly instantiable. — *end note*]

5 Directly instantiable types which derive, directly or indirectly, from the `surface` class shall either provide well-defined semantics for the graphics data graphics resource or inherit well-defined semantics for the graphics data graphics resource from a base class.

6 [*Example:* The `image_surface` class and the `display_surface` class each specify that they manage a raster graphics data graphics resource and that the members they inherit from the `surface` class shall use that raster graphics data graphics resource as their graphics data graphics resource. Since, unlike graphics data, raster graphics data provides well-defined semantics, these classes meet the requirements for being directly instantiable. — *end example*]

7 The definitions of the rendering and composing operations in 11.15.3 shall only be applicable when the graphics data graphics resource on which the `surface` members operate is a raster graphics data graphics resource. In all other cases, any attempt to invoke the rendering and composing operations shall result in undefined behavior.

11.15.2 surface synopsis [io2d.surface.synopsis]

```
namespace std::experimental::io2d::v1 {
  class surface {
  public:
    surface() = delete;

    // 11.15.9, state modifiers:
    void flush();
    void flush(error_code& ec) noexcept;
    void mark_dirty();
    void mark_dirty(error_code& ec) noexcept;
    void mark_dirty(const rectangle& rect);
    void mark_dirty(const rectangle& rect, error_code& ec) noexcept;
    void map(const function<void(mapped_surface&)>& action);
    void map(const function<void(mapped_surface&, error_code&)>& action,
      error_code& ec);
    void map(const function<void(mapped_surface&)>& action,
      const rectangle& extents);
    void map(const function<void(mapped_surface&, error_code&)>& action,
      const rectangle& extents, error_code& ec);

    // 11.15.10, render modifiers:
    void paint(const brush& b, const optional<brush_props>& bp = nullopt,
      const optional<render_props>& rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    template <class Allocator>
    void stroke(const brush& b, const path_builder<Allocator>& pb,
      const optional<brush_props>& bp = nullopt,
      const optional<stroke_props>& sp = nullopt,
      const optional<dashes>& d = nullopt,
      const optional<render_props>& rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    void stroke(const brush& b, const path_group& pg,
      const optional<brush_props>& bp = nullopt,
      const optional<stroke_props>& sp = nullopt,
      const optional<dashes>& d = nullopt,
      const optional<render_props>& rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    template <class Allocator>
    void fill(const brush& b, const path_builder<Allocator>& pb,
      const optional<brush_props>& bp = nullopt,
      const optional<render_props>& rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    void fill(const brush& b, const path_group& pg,
```

```
      const optional<brush_props>& bp = nullopt,
      const optional<render_props>& rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    template <class Allocator>
    void mask(const brush& b, const brush& mb,
      const optional<brush_props>& bp = nullopt,
      const optional<mask_props>& mp = nullopt,
      const optional<render_props>&rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    void mask(const brush& b, const brush& mb,
      const optional<brush_props>& bp = nullopt,
      const optional<mask_props>& mp = nullopt,
      const optional<render_props>&rp = nullopt,
      const optional<clip_props>& cl = nullopt);
  };
}
```

11.15.3 Rendering and composing [io2d.surface.rendering]

11.15.3.1 Operations [io2d.surface.rendering.ops]

1 The `surface` class provides four fundamental rendering and composing operations:

Table 18 — `surface` rendering and composing operations

Operation	Function(s)
Painting	`surface::paint`
Filling	`surface::fill`
Stroking	`surface::stroke`
Masking	`surface::mask`

2 All composing operations shall happen in a linear color space, regardless of the color space of the graphics data that is involved.

3 [*Note:* While a color space such as sRGB helps produce expected, consistent results when graphics data are viewed by people, composing operations only produce expected results when the channel data in the graphics data involved are uniformly (i.e. linearly) spaced. — *end note*]

11.15.3.2 Rendering and composing brushes [io2d.surface.rendering.brushes]

1 All rendering and composing operations use a *source brush* of type `brush`.

2 The masking operation uses a *mask brush* of type `brush`.

11.15.3.3 Rendering and composing source path [io2d.surface.rendering.sourcepath]

1 In addition to brushes (11.15.3.2), all rendering and composing operation except for painting use a *source path* of type `path_group`.

11.15.3.4 Common state data [io2d.surface.rendering.commonstate]

1 All rendering and composing operations use the following state data:

Table 19 — `surface` rendering and composing common state data

Name	Type
Brush properties	`brush_props`
Surface properties	`render_props`

Name	Type
Clip properties	`clip_props`

11.15.3.5 Specific state data [io2d.surface.rendering.specificstate]

1 In addition to the common state data (11.15.3.4), certain rendering and composing operations use state data that is specific to each of them:

Table 20 — `surface` rendering and composing specific state data

Operation	Name	Type
Stroking	Stroke properties	`stroke_props`
Stroking	Dashes	`dashes`
Masking	Mask properties	`mask_props`

11.15.3.6 State data default values [io2d.surface.rendering.statedefaults]

1 For all rendering and composing operations, the state data objects named above are provided using `optional<T>` class template arguments.

2 If there is no contained value for a state data object, it is interpreted as-if the `optional<T>` argument contained a default constructed object of the relevant state data object.

11.15.4 Standard coordinate spaces [io2d.surface.coordinatespaces]

1 There are four standard coordinate spaces relevant to the rendering and composing operations (11.15.3):

(1.1) — the brush coordinate space;

(1.2) — the mask coordinate space;

(1.3) — the user coordinate space; and

(1.4) — the surface coordinate space.

2 The *brush coordinate space* is the standard coordinate space of the source brush (11.15.3.2). Its transformation matrix is the brush properties' brush matrix (11.11.1).

3 The *mask coordinate space* is the standard coordinate space of the mask brush (11.15.3.2). Its transformation matrix is the mask properties' mask matrix (11.14.1).

4 The *user coordinate space* is the standard coordinate space of `path_group` objects. Its transformation matrix is a default-constructed `matrix_2d`.

5 The *surface coordinate space* is the standard coordinate space of the `surface` object's underlying graphics data graphics resource. Its transformation matrix is the surface properties' surface matrix (11.10.1).

6 Given a point `pt`, a brush coordinate space transformation matrix `bcsm`, a mask coordinate space transformation matrix `mcsm`, a user coordinate space transformation matrix `ucsm`, and a surface coordinate space transformation matrix `scsm`, the following table describes how to transform it from each of these standard coordinate spaces to the other standard coordinate spaces:

Table 21 — Point transformations

From	To	Transform
brush coordinate space	mask coordinate space	`mcsm.transform_point(bcsm.invert().transform_point(pt)).`
brush coordinate space	user coordinate space	`bcsm.invert().transform_point(pt).`

Table 21 — Point transformations (continued)

From	To	Transform
brush coordinate space	surface coordinate space	`scsm.transform_point(bcsm.invert().transform_point(pt)).`
user coordinate space	brush coordinate space	`bcsm.transform_point(pt).`
user coordinate space	mask coordinate space	`mcsm.transform_point(pt).`
user coordinate space	surface coordinate space	`scsm.transform_point(pt).`
surface coordinate space	brush coordinate space	`bcsm.transform_point(scsm.invert().transform_point(pt)).`
surface coordinate space	mask coordinate space	`mcsm.transform_point(scsm.invert().transform_point(pt)).`
surface coordinate space	user coordinate space	`scsm.invert().transform_point(pt).`

11.15.5 `surface` painting [io2d.surface.painting]

1 When a painting operation is initiated on a surface, the implementation shall produce results as-if the following steps were performed:

1. For each integral point *sp* of the underlying graphics data graphics resource, determine if *sp* is within the clip area (`clipprops.summary`); if so, proceed with the remaining steps.
2. Transform *sp* from the surface coordinate space (11.15.4) to the brush coordinate space (Table 21), resulting in point *bp*.
3. Sample from point *bp* of the source brush (11.15.3.2), combine the resulting visual data with the visual data at point *sp* in the underlying graphics data graphics resource in the manner specified by the surface's current *compositing operator* (11.10.1), and modify the visual data of the underlying graphics data graphics resource at point *sp* to reflect the result produced by application of the compositing operator.

11.15.6 `surface` filling [io2d.surface.filling]

1 When a filling operation is initiated on a surface, the implementation shall produce results as-if the following steps were performed:

1. For each integral point *sp* of the underlying graphics data graphics resource, determine if *sp* is within the *clip area* (11.12.1); if so, proceed with the remaining steps.
2. Transform *sp* from the surface coordinate space (11.15.4) to the user coordinate space (Table 21), resulting in point *up*.
3. Using the source path (11.15.3.3) and the fill rule (11.11.1), determine whether *up* shall be filled; if so, proceed with the remaining steps.
4. Transform *up* from the user coordinate space to the brush coordinate space (11.15.4 and Table 21), resulting in point *bp*.
5. Sample from point *bp* of the source brush (11.15.3.2), combine the resulting visual data with the visual data at point *sp* in the underlying graphics data graphics resource in the manner specified by the surface's current compositing operator (11.10.1), and modify the visual data of the underlying graphics data graphics resource at point *sp* to reflect the result produced by application of the compositing operator.

11.15.7 `surface` stroking [io2d.surface.stroking]

1 When a stroking operation is initiated on a surface, it is carried out for each path in the source path (11.15.3).

2 The following rules shall apply when a stroking operation is carried out on a path:

1. No part of the underlying graphics data graphics resource that is outside of the clip area shall be modified.
2. If the path only contains a degenerate path segment, then if the *line cap* value (see: 11.13.1 and 11.15.3.5) is either `line_cap::round` or `line_cap::square`, the line caps shall be rendered, resulting in a circle or a square, respectively. The remaining rules shall not apply.
3. If the path is a closed path, then the point where the end point of its final path segment meets the start point of the initial path segment shall be rendered as specified by the *line join* value (see: 11.13.1 and 11.15.3.5); otherwise the start point of the initial path segment and end point of the final path segment shall each by rendered as specified by the line cap value. The remaining meetings between successive end points and start points shall be rendered as specified by the line join value.
4. If the dash pattern (Table 20) has its default value or if its `vector<float>` member is empty, the path segments shall be rendered as a continuous path.
5. If the dash pattern's `vector<float>` member contains only one value, that value shall be used to define a repeating pattern in which the path is shown then hidden. The ends of each shown portion of the path shall be rendered as specified by the line cap value.
6. If the dash pattern's `vector<float>` member contains two or more values, the values shall be used to define a pattern in which the path is alternatively rendered then not rendered for the length specified by the value. The ends of each rendered portion of the path shall be rendered as specified by the line cap value. If the dash pattern's `float` member, which specifies an offset value, is not `0.0f`, the meaning of its value is implementation-defined. If a rendered portion of the path overlaps a not rendered portion of the path, the rendered portion shall be rendered.

3 When a stroking operation is carried out on a path, the width of each rendered portion shall be the *line width* (see: 11.13.1 and 11.15.3.5). Ideally this means that the diameter of the stroke at each rendered point should be equal to the line width. However, because there is an infinite number of points along each rendered portion, implementations may choose an unspecified method of determining minimum distances between points along each rendered portion and the diameter of the stroke between those points shall be the same. [*Note:* This concept is sometimes referred to as a tolerance. It allows for a balance between precision and performance, especially in situations where the end result is in a non-exact format such as raster graphics data. — *end note*]

4 After all paths in the path group have been rendered but before the rendered result is composed to the underlying graphics data graphics resource, the rendered result shall be transformed from the user coordinate space (11.15.4) to the surface coordinate space (11.15.4).

11.15.8 `surface` masking [io2d.surface.masking]

1 A *mask brush* is composed of a graphics data graphics resource, a `wrap_mode` value, a `filter` value, and a `matrix_2d` object.

2 When a masking operation is initiated on a surface, the implementation shall produce results as-if the following steps were performed:

1. For each integral point *sp* of the underlying graphics data graphics resource, determine if *sp* is within the clip area (11.12.1); if so, proceed with the remaining steps.
2. Transform *sp* from the surface coordinate space (11.15.4) to the mask coordinate space (Table 21), resulting in point *mp*.
3. Sample the alpha channel from point *mp* of the mask brush and store the result in *mac*; if the visual

data format of the mask brush does not have an alpha channel, the value of *mac* shall always be 1.0.

4. Transform *sp* from the surface coordinate space to the brush coordinate space, resulting in point *bp*.
5. Sample from point *bp* of the source brush (11.15.3.2), combine the resulting visual data with the visual data at point *sp* in the underlying graphics data graphics resource in the manner specified by the surface's current compositing operator (11.10.1), multiply each channel of the result produced by application of the compositing operator by *map* if the visual data format of the underlying graphics data graphics resource is a premultiplied format and if not then just multiply the alpha channel of the result by *map*, and modify the visual data of the underlying graphics data graphics resource at point *sp* to reflect the multiplied result.

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

```
void flush();
void flush(error_code& ec) noexcept;
```

1 *Effects:* If the implementation does not provide a native handle to the surface's underlying graphics data graphics resource, this function does nothing.

2 If the implementation does provide a native handle to the surface's underlying graphics data graphics resource, then the implementation performs every action necessary to ensure that all operations on the surface that produce observable effects occur.

3 The implementation performs any other actions necessary to ensure that the surface will be usable again after a call to `surface::mark_dirty`.

4 Once a call to `surface::flush` is made, `surface::mark_dirty` shall be called before any other member function of the surface is called or the surface is used as an argument to any other function.

5 *Throws:* As specified in Error reporting (4).

6 *Remarks:* This function exists to allow the user to take control of the underlying surface using an implementation-provided native handle without introducing a race condition. The implementation's responsibility is to ensure that the user can safely use the underlying surface.

7 *Error conditions:* The potential errors are implementation-defined.

8 Implementations should avoid producing errors here.

9 If the implementation does not provide a native handle to the `surface` object's underlying graphics data graphics resource, this function shall not produce any errors.

10 [*Note:* There are several purposes for `surface::flush` and `surface::mark_dirty`.

11 One is to allow implementation wide latitude in how they implement the rendering and composing operations (11.15.3), such as batching calls and then sending them to the underlying rendering and presentation technologies at appropriate times.

12 Another is to give implementations the chance during the call to `surface::flush` to save any internal state that might be modified by the user and then restore it during the call to `surface::mark_dirty`.

13 Other uses of this pair of calls are also possible. — *end note*]

```
void mark_dirty();
void mark_dirty(error_code& ec) noexcept;
void mark_dirty(const rectangle& extents);
void mark_dirty(const rectangle& extents, error_code& ec) noexcept;
```

14 *Effects:* If the implementation does not provide a native handle to the `surface` object's underlying graphics data graphics resource, this function shall do nothing.

15 If the implementation does provide a native handle to the `surface` object's underlying graphics data

graphics resource, then:

(15.1) — If called without a `rect` argument, informs the implementation that external changes using a native handle were potentially made to the entire underlying graphics data graphics resource.

(15.2) — If called with a `rect` argument, informs the implementation that external changes using a native handle were potentially made to the underlying graphics data graphics resource within the bounds specified by the *bounding rectangle* `rectangle{ round(extents.x()), round (extents.y()), round(extents.width()), round(extents.height())}`. No part of the bounding rectangle shall be outside of the bounds of the underlying graphics data graphics resource; no diagnostic is required.

16 *Throws:* As specified in Error reporting (4).

17 *Remarks:* After external changes are made to this `surface` object's underlying graphics data graphics resource using a native pointer, this function shall be called before using this `surface` object; no diagnostic is required.

18 No call to this function shall be required solely as a result of changes made to a surface using the functionality provided by `surface::map`. [*Note:* The `mapped_surface` type, which is used by `surface::map`, provides its own functionality for managing any such changes. — *end note*]

19 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

20 If the implementation does not provide a native handle to the `surface` object's underlying graphics data graphics resource, this function shall not produce any errors.

```
void map(const function<void(mapped_surface&)>& action);
void map(const function<void(mapped_surface&, error_code&)>& action, error_code& ec);
void map(const function<void(mapped_surface&)>& action, const rectangle& extents);
void map(const function<void(mapped_surface&, error_code&)>& action,
  const rectangle& extents, error_code& ec);
```

21 *Effects:* Creates a `mapped_surface` object and calls `action` using it.

22 The `mapped_surface` object is created using `*this`, which allows direct manipulation of the underlying graphics data graphics resource.

23 If called with a `const rectangle& extents` argument, the `mapped_surface` object shall only allow manipulation of the portion of `*this` specified by the *bounding rectangle* `rectangle{ round(extents.x()), round(extents.y()), round(extents.width()), round(extents.height())}`. If any part of the bounding rectangle is outside of the bounds of `*this`, the call shall result in undefined behavior; no diagnostic is required.

24 *Throws:* As specified in Error reporting (4).

25 *Remarks:* Whether changes are committed to the underlying graphics data graphics resource immediately or only when the `mapped_surface` object is destroyed is unspecified.

26 Calling this function on a `surface` object and then calling any function on the `surface` object or using the `surface` object before the call to this function has returned shall result in undefined behavior; no diagnostic is required.

27 *Error conditions:* `errc::not_supported` if a `mapped_surface` object cannot be created for the `surface` object. The `surface` object is not modified if an error occurs.

11.15.10 surface render modifiers [io2d.surface.modifiers.render]

```
void paint(const brush& b, const optional<brush_props>& bp = nullopt,
  const optional<render_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

1 *Effects:* Performs the painting rendering and composing operation as specified by 11.15.5.

2 The meanings of the parameters are specified by 11.15.3.

3 *Throws:* As specified in Error reporting (4).

4 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
void stroke(const brush& b, const path_builder<Allocator>& pb,
  const optional<brush_props>& bp = nullopt,
  const optional<stroke_props>& sp = nullopt,
  const optional<dashes>& d = nullopt,
  const optional<render_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
void stroke(const brush& b, const path_group& pg,
  const optional<brush_props>& bp = nullopt,
  const optional<stroke_props>& sp = nullopt,
  const optional<dashes>& d = nullopt,
  const optional<render_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

5 *Effects:* Performs the stroking rendering and composing operation as specified by 11.15.7.

6 The meanings of the parameters are specified by 11.15.3.

7 *Throws:* As specified in Error reporting (4).

8 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
void fill(const brush& b, const path_builder<Allocator>& pb,
  const optional<brush_props>& bp = nullopt,
  const optional<render_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
void fill(const brush& b, const path_group& pg,
  const optional<brush_props>& bp = nullopt,
  const optional<render_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

9 *Effects:* Performs the filling rendering and composing operation as specified by 11.15.6.

10 The meanings of the parameters are specified by 11.15.3.

11 *Throws:* As specified in Error reporting (4).

12 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
void mask(const brush& b, const brush& mb,
  const path_builder<Allocator>& pb,
  const optional<brush_props>& bp = nullopt,
  const optional<mask_props>& mp = nullopt,
  const optional<render_props>&rp = nullopt,
  const optional<clip_props>& cl = nullopt);
void mask(const brush& b, const brush& mb, const path_group& pg,
  const optional<brush_props>& bp = nullopt,
  const optional<mask_props>& mp = nullopt,
  const optional<render_props>&rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

13 *Effects:* Performs the masking rendering and composing operation as specified by 11.15.8.

14 The meanings of the parameters are specified by 11.15.3.

15 *Throws:* As specified in Error reporting (4).

16 *Error conditions:*

The errors, if any, produced by this function are implementation-defined.

11.16 Class `image_surface` [io2d.imagesurface]

11.16.1 `image_surface` summary [io2d.imagesurface.summary]

1 The class `image_surface` derives from the `surface` class and provides an interface to a raster graphics data graphics resource.

2 [*Note:* Because of the functionality it provides and what it can be used for, it is expected that developers familiar with other graphics technologies will think of the `image_surface` class as being a form of *render target*. This is intentional, though this Technical Specification does not formally define or use that term to avoid any minor ambiguities and differences in its meaning between the various graphics technologies that do use the term render target. — *end note*]

11.16.2 `image_surface` synopsis [io2d.imagesurface.synopsis]

```
namespace std::experimental::io2d::v1 {
  class image_surface : public surface {
  public:
    // 11.16.3, construct/copy/move/destroy:
    image_surface() = delete;
    image_surface(experimental::io2d::format fmt, int width, int height);
    image_surface(filesystem::path f, image_file_format i,
      experimental::io2d::format fmt);

    // 11.16.4, members:
    void save(filesystem::path p, image_file_format i);

    // 11.16.5, observers:
    experimental::io2d::format format() const noexcept;
    int width() const noexcept;
    int height() const noexcept;
  };
}
```

11.16.3 `image_surface` constructors and assignment operators [io2d.imagesurface.cons]

```
image_surface(experimental::io2d::format fmt, int width, int height);
```

1 *Requires:* `w >= 1`.

2 `h >= 1`.

3 *Effects:* Constructs an object of type `image_surface`.

4 *Postconditions:* `this->format() == fmt`.

5 `this->width() == width`.

6 `this->height() == height`.

```
image_surface(filesystem::path f, image_file_format i,
  experimental::io2d::format fmt);
```

7 *Requires:* `f` is a file and its contents are data in either JPEG format or PNG format.

8 *Effects:* Constructs an object of type `image_surface`.

9 The data of the underlying raster graphics data graphics resource is the raster graphics data that results from processing `f` into uncompressed raster graphics in the manner specified by the standard that specifies how to transform the contents of data contained in `f` into raster graphics data and then transforming that raster graphics data into the format specified by `fmt`.

10 The data of `f` is processed into uncompressed raster graphics data as specified by the value of `i`.

11 If `i` is `image_file_format::unknown`, it is implementation-defined whether the surface is created in the image file format, if any, that the implementation associates with `p.extension()` provided that `p.has_extension() == true`. If `p.has_extension() == false`, the implementation does not associate an image file format with `p.extension()`, or the implementation does not support reading in graphics data in that image file format, the error specified below occurs.

12 The resulting uncompressed raster graphics data is then transformed into the data format specified by `fmt`. If the format specified by `fmt` only contains an alpha channel, the values of the color channels, if any, of the underlying raster graphics data graphics resource are unspecified. If the format specified by `fmt` only contains color channels and the resulting uncompressed raster graphics data is in a premultiplied format, then the value of each color channel for each pixel shall be divided by the value of the alpha channel for that pixel. The visual data shall then be set as the visual data of the underlying raster graphics data graphics resource.

13 *Throws:* As specified in Error reporting [io2d.err.report].

14 *Error conditions:* Any error that could result from trying to access `f`, open `f` for reading, or reading data from `f`.

15 `errc::not_supported` if `image_file_format::unknown` is passed as an argument and the implementation is unable to determine the file format or does not support saving in the image file format it determined.

16 Other errors, if any, produced by this function are implementation-defined.

11.16.4 `image_surface` members [io2d.imagesurface.members]

```
void save(filesystem::path p, image_file_format i);
```

1 *Requires:* `p` shall be a valid path to a file. The file need not exist provided that the other components of the path are valid.

2 If the file exists, it shall be writable. If the file does not exist, it shall be possible to create the file at the specified path and then the created file shall be writable.

3 *Effects:* Any pending rendering and composing operations (11.15.3) are performed.

4 The visual data of the underlying raster graphics data graphics resource is written to `p` in the data format specified by `i`.

5 If `i` is `image_file_format::unknown`, it is implementation-defined whether the surface is saved in the image file format, if any, that the implementation associates with `p.extension()` provided that `p.has_extension() == true`. If `p.has_extension() == false`, the implementation does not associate an image file format with `p.extension()`, or the implementation does not support saving in that image file format, the error specified below occurs.

6 *Throws:* As specified in Error reporting [io2d.err.report].

7 *Error conditions:* Any error that could result from trying to create `f`, access `f`, or write data to `f`.

8 `errc::not_supported` if `image_file_format::unknown` is passed as an argument and the implementation is unable to determine the file format or does not support saving in the image file format it

determined.

9 Other errors, if any, produced by this function are implementation-defined.

11.16.5 `image_surface` observers [io2d.imagesurface.observers]

`experimental::io2d::format format() const noexcept;`

1 *Returns:* The pixel format of the `image_surface` object.

2 *Remarks:* If the `image_surface` object is invalid, this function shall return `experimental::io2d::format::invalid`.

`int width() const noexcept;`

3 *Returns:* The number of pixels per horizontal line of the `image_surface` object.

4 *Remarks:* This function shall return the value `0` if `this->format() == experimental::io2d::format::invalid`.

`int height() const noexcept;`

5 *Returns:* The number of horizontal lines of pixels in the `image_surface` object.

6 *Remarks:* This function shall return the value `0` if `this->format() == experimental::io2d::format::invalid`.

11.17 Class `display_surface` [io2d.displaysurface]

11.17.1 `display_surface` description [io2d.displaysurface.intro]

1 The class `display_surface` derives from the `surface` class and provides an interface to a pixmap called the *back buffer* and to a second pixmap called the *display buffer*.

2 The pixel data of the display buffer can never be accessed by the user except through a native handle, if one is provided. As such, its pixel format need not equate to any of the pixel formats described by the `experimental::io2d::format` enumerators. This is meant to give implementors more flexibility in trying to display the pixels of the back buffer in a way that is visually as close as possible to the colors of those pixels.

3 The *draw callback* (Table 22) is called by `display_surface::show` as required by the refresh rate and when otherwise needed by the implementation in order to update the pixel content of the back buffer.

4 After each execution of the draw callback, the contents of the back buffer are transferred using sampling with an unspecified filter to the display buffer. The display buffer is then shown to the user via the *output device*. [*Note:* The filter is unspecified to allow implementations to achieve the best possible result, including by changing filters at runtime depending on factors such as whether scaling is required and by using specialty hardware if available, while maintaining a balance between quality and performance that the implementer deems acceptable.

In the absence of specialty hardware, implementers are encouraged to use a filter that is the equivalent of a nearest neighbor interpolation filter if no scaling is required and otherwise to use a filter that produces results that are at least as good as those that would be obtained by using a bilinear interpolation filter. — *end note*]

11.17.2 `display_surface` synopsis [io2d.displaysurface.synopsis]

```
namespace std::experimental::io2d::v1 {
  class display_surface : public surface {
  public:
    // 11.17.5, construct/copy/move/destroy:
    display_surface(display_surface&& other) noexcept;
    display_surface& operator=(display_surface&& other) noexcept;
```

```
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
  experimental::io2d::refresh_rate rr =
  experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f);
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat, error_code& ec,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
  experimental::io2d::refresh_rate rr =
  experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f)
  noexcept;

display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  int preferredDisplayWidth, int preferredDisplayHeight,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
  experimental::io2d::refresh_rate rr =
  experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f);
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  int preferredDisplayWidth, int preferredDisplayHeight, error_code& ec,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
  experimental::io2d::refresh_rate rr =
  experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f)
  noexcept;

~display_surface();

// 11.17.6, modifiers:
void draw_callback(const function<void(display_surface& sfc)>& fn) noexcept;
void size_change_callback(const function<void(display_surface& sfc)>& fn)
  noexcept;
void width(int w);
void width(int w, error_code& ec) noexcept;
void height(int h);
void height(int h, error_code& ec) noexcept;
void display_width(int w);
void display_width(int w, error_code& ec) noexcept;
void display_height(int h);
void display_height(int h, error_code& ec) noexcept;
void dimensions(int w, int h);
void dimensions(int w, int h, error_code& ec) noexcept;
void display_dimensions(int dw, int dh);
void display_dimensions(int dw, int dh, error_code& ec) noexcept;
void scaling(experimental::io2d::scaling scl) noexcept;
void user_scaling_callback(const function<experimental::io2d::rectangle(
  const display_surface&, bool&)>& fn) noexcept;
void letterbox_brush(const optional<brush>& b,
  const optional<brush_props> = nullopt) noexcept;
void auto_clear(bool val) noexcept;
void refresh_rate(experimental::io2d::refresh_rate rr) noexcept;
bool desired_frame_rate(float fps) noexcept;
void redraw_required() noexcept;
int begin_show();
```

```
    void end_show();

    // 11.17.7, observers:
    experimental::io2d::format format() const noexcept;
    int width() const noexcept;
    int height() const noexcept;
    int display_width() const noexcept;
    int display_height() const noexcept;
    vector_2d dimensions() const noexcept;
    vector_2d display_dimensions() const noexcept;
    experimental::io2d::scaling scaling() const noexcept;
    function<experimental::io2d::rectangle(const display_surface&,
      bool&)> user_scaling_callback() const;
    function<experimental::io2d::rectangle(const display_surface&,
      bool&)> user_scaling_callback(error_code& ec) const noexcept;
    optional<brush> letterbox_brush() const noexcept;
    bool auto_clear() const noexcept;
    experimental::io2d::refresh_rate refresh_rate() const noexcept;
    float desired_frame_rate() const noexcept;
    float elapsed_draw_time() const noexcept;
  };
}
```

11.17.3 `display_surface` miscellaneous behavior [io2d.displaysurface.misc]

1 What constitutes an output device is implementation-defined, with the sole constraint being that an output device must allow the user to see the dynamically-updated contents of the display buffer. [*Example:* An output device might be a window in a windowing system environment or the usable screen area of a smart phone or tablet. — *end example*]

2 Implementations do not need to support the simultaneous existence of multiple `display_surface` objects.

3 All functions inherited from `surface` that affect its underlying graphics data graphics resource shall operate on the back buffer.

11.17.4 `display_surface` state [io2d.displaysurface.state]

1 Table 22 specifies the name, type, function, and default value for each item of a display surface's observable state.

Table 22 — Display surface observable state

Name	Type	Function	Default value
Letterbox brush	`brush`	This is the brush that shall be used as specified by `scaling::letterbox` (Table 15)	`brush{ { rgba_color::black() } }`
Letterbox brush props	`brush_props`	This is the brush properties for the letterbox brush	`brush_props{ }`
Scaling type	`scaling`	When the user scaling callback is equal to its default value, this is the type of scaling that shall be used when transferring the back buffer to the display buffer	`scaling::letterbox`

Table 22 — Display surface observable state (continued)

Name	Type	Function	Default value
Draw width	`int`	The width in pixels of the back buffer. The minimum value is `1`. The maximum value is unspecified. Because users can only request a preferred value for the draw width when setting and altering it, the maximum value may be a run-time determined value. If the preferred draw width exceeds the maximum value, then if a preferred draw height has also been supplied then implementations should provide a back buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred draw width and the preferred draw height otherwise implementations should provide a back buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred draw width and the current draw height	*N/A* [*Note:* It is impossible to create a `display_surface` object without providing a preferred draw width value; as such a default value cannot exist. — *end note*]

Table 22 — Display surface observable state (continued)

Name	Type	Function	Default value
Draw height	`int`	The height in pixels of the back buffer. The minimum value is `1`. The maximum value is unspecified. Because users can only request a preferred value for the draw height when setting and altering it, the maximum value may be a run-time determined value. If the preferred draw height exceeds the maximum value, then if a preferred draw width has also been supplied then implementations should provide a back buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred draw width and the preferred draw height otherwise implementations should provide a back buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the current draw width and the preferred draw height	*N/A* [*Note:* It is impossible to create a `display_surface` object without providing a preferred draw height value; as such a default value cannot exist. — *end note*]
Draw format	`format`	The pixel format of the back buffer. When a `display_surface` object is created, a preferred pixel format value is provided. If the implementation does not support the preferred pixel format value as the value of draw format, the resulting value of draw format is implementation-defined	*N/A* [*Note:* It is impossible to create a `display_surface` object without providing a preferred draw format value; as such a default value cannot exist. — *end note*]

Table 22 — Display surface observable state (continued)

Name	Type	Function	Default value
Display width	`int`	The width in pixels of the display buffer. The minimum value is unspecified. The maximum value is unspecified. Because users can only request a preferred value for the display width when setting and altering it, both the minimum value and the maximum value may be run-time determined values. If the preferred display width is not within the range between the minimum value and the maximum value, inclusive, then if a preferred display height has also been supplied then implementations should provide a display buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred display width and the preferred display height otherwise implementations should provide a display buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred display width and the current display height	*N/A* [*Note:* It is impossible to create a `display_surface` object without providing a preferred display width value since in the absence of an explicit display width argument the mandatory preferred draw width argument is used as the preferred display width; as such a default value cannot exist. — *end note*]

Table 22 — Display surface observable state (continued)

Name	Type	Function	Default value
Display height	`int`	The height in pixels of the display buffer. The minimum value is unspecified. The maximum value is unspecified. Because users can only request a preferred value for the display height when setting and altering it, both the minimum value and the maximum value may be run-time determined values. If the preferred display height is not within the range between the minimum value and the maximum value, inclusive, then if a preferred display width has also been supplied then implementations should provide a display buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred display width and the preferred display height otherwise implementations should provide a display buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the current display width and the preferred display height	*N/A* [*Note:* It is impossible to create a `display_surface` object without providing a preferred display height value since in the absence of an explicit display height argument the mandatory preferred draw height argument is used as the preferred display height; as such a default value cannot exist. — *end note*]
Draw callback	`function< void( display_-surface&)>`	This function shall be called in a continuous loop when `display_surface::show` is executing. It is used to draw to the back buffer, which in turn results in the display of the drawn content to the user	`nullptr`

Table 22 — Display surface observable state (continued)

Name	Type	Function	Default value
Size change callback	`function< void( display_surface&)>`	If it exists, this function shall be called whenever the display buffer has been resized. Neither the display width nor the display height shall be changed by the size change callback; no diagnostic is required [*Note:* This means that there has been a change to the display width, display height, or both. Its intent is to allow the user the opportunity to change other observable state, such as the draw width, draw height, or scaling type, in reaction to the change. — *end note*]	`nullptr`
User scaling callback	`function< rectangle( const display_surface&, bool&)>`	If it exists, this function shall be called whenever the contents of the back buffer need to be copied to the display buffer. The function is called with the const reference to `display_surface` object and a reference to a `bool` variable which has the value `false`. If the value of the `bool` is `true` when the function returns, the letterbox brush shall be used as specified by `scaling::letterbox` (Table 15). The function shall return a `rectangle` object that defines the area within the display buffer to which the back buffer shall be transferred. The `rectangle` may include areas outside of the bounds of the display buffer, in which case only the area of the back buffer that lies within the bounds of the display buffer will ultimately be visible to the user	`nullptr`

Table 22 — Display surface observable state (continued)

Name	Type	Function	Default value
Auto clear	`bool`	If `true` the implementation shall call `surface::clear`, which shall clear the back buffer, immediately before it executes the draw callback	`false`
Refresh rate	`refresh_rate`	The `refresh_rate` value that determines when the draw callback shall be called while `display_surface::show` is being executed	`refresh_rate::as_fast_as_possible`
Desired frame rate	`float`	This value is the number of times the draw callback shall be called per second while `display_surface::show` is being executed when the value of refresh rate is `refresh_rate::fixed`, subject to the additional requirements documented in the meaning of `refresh_rate::fixed` (Table 16)	

11.17.5 `display_surface` constructors and assignment operators [io2d.displaysurface.cons]

```
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
  experimental::io2d::refresh_rate rr =
  experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f);
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat, error_code& ec,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
  experimental::io2d::refresh_rate rr =
  experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f)
  noexcept;
```

1 *Requires:* `preferredWidth > 0`.

2 `preferredHeight > 0`.

3 `preferredFormat != experimental::io2d::format::invalid`.

4 *Effects:* Constructs an object of type `display_surface`.

5 The preferredWidth parameter specifies the preferred width value for draw width and display width. The preferredHeight parameter specifies the preferred height value for draw height and display height. draw width and display width need not have the same value. draw height and display height need not have the same value.

6 The preferredFormat parameter specifies the preferred pixel format value for draw format.

7 The value of scaling type shall be the value of `scl`.

8 The value of refresh rate shall be the value of `rr`.

9 The value of desired frame rate shall be as-if `display_surface::desired_frame_rate` was called with `fps` as its argument. If `!is_finite(fps)`, then the value of desired frame rate shall be its default value.

10 All other observable state data shall have their default values.

11 *Throws:* As specified in Error reporting (4).

12 *Error conditions:* `errc::not_supported` if creating the `display_surface` object would exceed the maximum number of simultaneous `display_surface` objects the implementation supports.

```
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  int preferredDisplayWidth, int preferredDisplayHeight,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
    experimental::io2d::refresh_rate rr =
    experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f);
display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  int preferredDisplayWidth, int preferredDisplayHeight, error_code& ec,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox,
    experimental::io2d::refresh_rate rr =
    experimental::io2d::refresh_rate::as_fast_as_possible, float fps = 30.0f)
  noexcept;
```

13 *Requires:* `preferredWidth > 0`.

14 `preferredHeight > 0`.

15 `preferredDisplayWidth > 0`.

16 `preferredDisplayHeight > 0`.

17 `preferredFormat != experimental::io2d::format::invalid`.

18 *Effects:* Constructs an object of type `display_surface`.

19 The preferredWidth parameter specifies the preferred width value for draw width. The preferredDisplayWidth parameter specifies the preferred display width value for display width. The preferredHeight parameter specifies the preferred height value for draw height. The preferredDisplayHeight parameter specifies the preferred display height value for display height.

20 The preferredFormat parameter specifies the preferred pixel format value for draw format.

21 The value of scaling type shall be the value of `scl`.

22 The value of refresh rate shall be the value of `rr`.

23 The value of desired frame rate shall be as-if `display_surface::desired_frame_rate` was called with `fps` as its argument. If `!is_finite(fps)`, then the value of desired frame rate shall be its default value.

24 All other observable state data shall have their default values.

25 *Throws:* As specified in Error reporting (4).

26 *Error conditions:* `errc::not_supported` if creating the `display_surface` object would exceed the maximum number of simultaneous `display_surface` objects the implementation supports.

11.17.6 `display_surface` modifiers [io2d.displaysurface.modifiers]

```
void draw_callback(const function<void(display_surface& sfc)>& fn) noexcept;
```

1 *Effects:* Sets the draw callback to `fn`.

```
void size_change_callback(const function<void(display_surface& sfc)>& fn)
  noexcept;
```

2 *Effects:* Sets the size change callback to `fn`.

```
void width(int w);
void width(int w, error_code& ec) noexcept;
```

3 *Effects:* If the value of draw width is the same as `w`, this function does nothing.

4 Otherwise, draw width is set as specified by Table 22 with `w` treated as being the preferred draw width.

5 If the value of draw width changes as a result, the implementation shall attempt to create a new back buffer with the updated dimensions while retaining the existing back buffer. The implementation may destroy the existing back buffer prior to creating a new back buffer with the updated dimensions only if it can guarantee that in doing so it will either succeed in creating the new back buffer or will be able to create a back buffer with the previous dimensions in the event of failure.

6 [*Note:* The intent of the previous paragraph is to ensure that, no matter the result, a valid back buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new back buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing back buffer even if they cannot determine in advance that creating the new back buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a back buffer with the previous dimensions. Regardless, there must be a valid back buffer when this call completes. — *end note*]

7 The value of the back buffer's pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

8 If an error occurs, the implementation shall ensure that the back buffer is valid and has the same dimensions it had prior to this call and that draw width shall retain its value prior to this call.

9 *Throws:* As specified in Error reporting (4).

10 *Error conditions:* `errc::invalid_argument` if `w <= 0` or if the value of `w` is greater than the maximum value for draw width.

`errc::not_enough_memory` if there is insufficient memory to create a back buffer with the updated dimensions.

Other errors, if any, produced by this function are implementation-defined.

```
void height(int h);
void height(int h, error_code& ec) noexcept;
```

11 *Effects:* If the value of draw height is the same as `h`, this function does nothing.

12 Otherwise, draw height is set as specified by Table 22 with `h` treated as being the preferred draw height.

13 If the value of draw height changes as a result, the implementation shall attempt to create a new back buffer with the updated dimensions while retaining the existing back buffer. The implementation may destroy the existing back buffer prior to creating a new back buffer with the updated dimensions only if it can guarantee that in doing so it will either succeed in creating the new back buffer or will be able to create a back buffer with the previous dimensions in the event of failure.

14 [*Note:* The intent of the previous paragraph is to ensure that, no matter the result, a valid back buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new back buffer successfully the previous one must be destroyed. The

previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing back buffer even if they cannot determine in advance that creating the new back buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a back buffer with the previous dimensions. Regardless, there must be a valid back buffer when this call completes. — *end note*]

15 The value of the back buffer's pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

16 If an error occurs, the implementation shall ensure that the back buffer is valid and has the same dimensions it had prior to this call and that draw height shall retain its value prior to this call.

17 *Throws:* As specified in Error reporting (4).

18 *Error conditions:* `errc::invalid_argument` if `h <= 0` or if the value of `h` is greater than the maximum value for draw height.

`errc::not_enough_memory` if there is insufficient memory to create a back buffer with the updated dimensions.

Other errors, if any, produced by this function are implementation-defined.

```
void display_width(int w);
void display_width(int w, error_code& ec) noexcept;
```

19 *Effects:* If the value of display width is the same as `w`, this function does nothing.

20 Otherwise, display width is set as specified by Table 22 with `w` treated as being the preferred display width.

21 If the value of display width changes as a result, the implementation shall attempt to create a new display buffer with the updated dimensions while retaining the existing display buffer. The implementation may destroy the existing display buffer prior to creating a new display buffer with the updated dimensions only if it can guarantee that in doing so it will either succeed in creating the new display buffer or will be able to create a display buffer with the previous dimensions in the event of failure.

22 [*Note:* The intent of the previous paragraph is to ensure that, no matter the result, a valid display buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new display buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing display buffer even if they cannot determine in advance that creating the new display buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a display buffer with the previous dimensions. Regardless, there must be a valid display buffer when this call completes. — *end note*]

23 The value of the display buffer's pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

24 If an error occurs, the implementation shall ensure that the display buffer is valid and has the same dimensions it had prior to this call and that display width shall retain its value prior to this call.

25 *Throws:* As specified in Error reporting (4).

26 *Error conditions:* `errc::invalid_argument` if the value of `w` is less than the minimum value for display width or if the value of `w` is greater than the maximum value for display width.

`errc::not_enough_memory` if there is insufficient memory to create a display buffer with the updated dimensions.

Other errors, if any, produced by this function are implementation-defined.

```
void display_height(int h);
```

```
void display_height(int h, error_code& ec) noexcept;
```

27 *Effects:* If the value of display height is the same as `h`, this function does nothing.

28 Otherwise, display height is set as specified by Table 22 with `h` treated as being the preferred display height.

29 If the value of display height changes as a result, the implementation shall attempt to create a new display buffer with the updated dimensions while retaining the existing display buffer. The implementation may destroy the existing display buffer prior to creating a new display buffer with the updated dimensions only if it can guarantee that in doing so it will either succeed in creating the new display buffer or will be able to create a display buffer with the previous dimensions in the event of failure.

30 [*Note:* The intent of the previous paragraph is to ensure that, no matter the result, a valid display buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new display buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing display buffer even if they cannot determine in advance that creating the new display buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a display buffer with the previous dimensions. Regardless, there must be a valid display buffer when this call completes. — *end note*]

31 The value of the display buffer's pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

32 If an error occurs, the implementation shall ensure that the display buffer is valid and has the same dimensions it had prior to this call and that display height shall retain its value prior to this call.

33 *Throws:* As specified in Error reporting (4).

34 *Error conditions:* `errc::invalid_argument` if the value of `h` is less than the minimum value for display height or if the value of `h` is greater than the maximum value for display height.

`errc::not_enough_memory` if there is insufficient memory to create a display buffer with the updated dimensions.

Other errors, if any, produced by this function are implementation-defined.

```
void dimensions(int w, int h);
void dimensions(int w, int h, error_code& ec) noexcept;
```

35 *Effects:* If the value of draw width is the same as `w` and the value of draw height is the same as `h`, this function does nothing.

36 Otherwise, draw width is set as specified by Table 22 with `w` treated as being the preferred draw width and draw height is set as specified by Table 22 with `h` treated as being the preferred draw height.

37 If the value of draw width changes as a result or the value of draw height changes as a result, the implementation shall attempt to create a new back buffer with the updated dimensions while retaining the existing back buffer. The implementation may destroy the existing back buffer prior to creating a new back buffer with the updated dimensions only if it can guarantee that in doing so it will either succeed in creating the new back buffer or will be able to create a back buffer with the previous dimensions in the event of failure.

38 [*Note:* The intent of the previous paragraph is to ensure that, no matter the result, a valid back buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new back buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing back buffer even if they cannot determine in advance that creating the new back buffer will succeed, provided that they can guarantee that if the attempt fails they can always

successfully recreate a back buffer with the previous dimensions. Regardless, there must be a valid back buffer when this call completes. — *end note*]

39 The value of the back buffer's pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

40 If an error occurs, the implementation shall ensure that the back buffer is valid and has the same dimensions it had prior to this call and that draw width and draw height shall retain the values they had prior to this call.

41 *Throws:* As specified in Error reporting (4).

42 *Error conditions:* `errc::invalid_argument` if `w <= 0`, if the value of `w` is greater than the maximum value for draw width, if `h <= 0` or if the value of `h` is greater than the maximum value for draw height.

`errc::not_enough_memory` if there is insufficient memory to create a back buffer with the updated dimensions.

Other errors, if any, produced by this function are implementation-defined.

```
void display_dimensions(int dw, int dh);
void display_dimensions(int dw, int dh, error_code& ec) noexcept;
```

43 *Effects:* If the value of display width is the same as `w` and the value of display height is the same as `h`, this function does nothing.

44 Otherwise, display width is set as specified by Table 22 with `w` treated as being the preferred display height and display height is set as specified by Table 22 with `h` treated as being the preferred display height.

45 If the value of display width or the value of display height changes as a result, the implementation shall attempt to create a new display buffer with the updated dimensions while retaining the existing display buffer. The implementation may destroy the existing display buffer prior to creating a new display buffer with the updated dimensions only if it can guarantee that in doing so it will either succeed in creating the new display buffer or will be able to create a display buffer with the previous dimensions in the event of failure.

46 [*Note:* The intent of the previous paragraph is to ensure that, no matter the result, a valid display buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new display buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing display buffer even if they cannot determine in advance that creating the new display buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a display buffer with the previous dimensions. Regardless, there must be a valid display buffer when this call completes. — *end note*]

47 If an error occurs, the implementation shall ensure that the display buffer is valid and has the same dimensions it had prior to this call and that display width and display height shall retain the values they had prior to this call.

48 If the display buffer has changed, even if its width and height have not changed, the draw callback shall be called.

49 If the width or height of the display buffer has changed, the size change callback shall be called if it's value is not its default value.

50 *Throws:* As specified in Error reporting (4).

51 *Error conditions:* `errc::invalid_argument` if the value of `w` is less than the minimum value for display width, if the value of `w` is greater than the maximum value for display width, if the value of `h` is less than the minimum value for display height, or if the value of `h` is greater than the maximum value for

display height.

`errc::not_enough_memory` if there is insufficient memory to create a display buffer with the updated dimensions.

Other errors, if any, produced by this function are implementation-defined.

```
void scaling(experimental::io2d::scaling scl) noexcept;
```

52 *Effects:* Sets scaling type to the value of `scl`.

```
void user_scaling_callback(const function<experimental::io2d::rectangle(
  const display_surface&, bool&)>& fn) noexcept;
```

53 *Effects:* Sets the user scaling callback to `fn`.

```
void letterbox_brush(const optional<brush&>b,
  const optional<brush_props>& bp = nullopt);
void letterbox_brush(const optional<brush&>b, error_code& ec,
  const optional<brush_props>& bp = nullopt) noexcept;
```

54 *Effects:* Sets the letterbox brush to the value contained in `b` if it contains a value, otherwise set letterbox brush to its default value.

55 Sets the letterbox brush props to the value contained in `bp` if it contains a value, otherwise sets it letterbox brush props to its default value.

56 *Throws:* As specified in Error reporting (4).

57 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void auto_clear(bool val) noexcept;
```

58 *Effects:* Sets auto clear to the value of `val`.

```
void refresh_rate(experimental::io2d::refresh_rate rr) noexcept;
```

59 *Effects:* Sets the refresh rate to the value of `rr`.

```
bool desired_frame_rate(float fps) noexcept;
```

60 *Effects:* If `!is_finite(fps)`, this function has no effects.

61 Sets the desired frame rate to an implementation-defined minimum frame rate if `fps` is less than the minimum frame rate, an implementation-defined maximum frame rate if `fps` is greater than the maximum frame rate, otherwise to the value of `fps`.

62 *Returns:* `false` if the desired frame rate was set to the value of `fps`; otherwise `true`.

```
void redraw_required() noexcept;
```

63 *Effects:* When `display_surface::begin_show` is executing, informs the implementation that the draw callback must be called as soon as possible.

```
int begin_show();
```

64 *Effects:* Performs the following actions in a continuous loop:

1. Handle any implementation and host environment matters. If there are no pending implementation or host environment matters to handle, proceed immediately to the next action.
2. Run the size change callback if doing so is required by its specification and it does not have a value equivalent to its default value.
3. If the refresh rate requires that the draw callback be called then:

a) Evaluate auto clear and perform the actions required by its specification, if any.

b) Run the draw callback.

c) Ensure that all operations from the draw callback that can effect the back buffer have completed.

d) Transfer the contents of the back buffer to the display buffer using sampling with an unspecified filter. If the user scaling callback does not have a value equivalent to its default value, use it to determine the position where the contents of the back buffer shall be transferred to and whether or not the letterbox brush should be used. Otherwise use the value of scaling type to determine the position and whether the letterbox brush should be used.

65 If `display_surface::end_show` is called from the draw callback, the implementation shall finish executing the draw callback and shall immediately cease to perform any actions in the continuous loop other than handling any implementation and host environment matters needed to exit the loop properly.

66 No later than when this function returns, the output device shall cease to display the contents of the display buffer.

67 What the output device shall display when it is not displaying the contents of the display buffer is unspecified.

68 *Returns:* The possible values and meanings of the possible values returned are implementation-defined.

69 *Throws:* As specified in Error reporting (4).

70 *Remarks:* Since this function calls the draw callback and can call the size change callback and the user scaling callback, in addition to the errors documented below, any errors that the callback functions produce can also occur.

71 *Error conditions:* 72 `errc::operation_would_block` if the value of draw callback is equivalent to its default value or if it becomes equivalent to its default value before this function returns.

73 Other errors, if any, produced by this function are implementation-defined.

```
void end_show();
```

74 *Effects:* If this function is called outside of the draw callback while it is being executed in the `display_surface::begin_show` function's continuous loop, it does nothing.

75 Otherwise, the implementation initiates the process of exiting the `display_surface::begin_show` function's continuous loop.

76 If possible, any procedures that the host environment requires in order to cause the `display_-surface::show` function's continuous loop to stop executing without error should be followed.

77 The `display_surface::begin_show` function's loop continues execution until it returns.

11.17.7 `display_surface` observers [io2d.displaysurface.observers]

```
experimental::io2d::format format() const noexcept;
```

1 *Returns:* The value of draw format.

```
int width() const noexcept;
```

2 *Returns:* The draw width.

```
int height() const noexcept;
```

3 *Returns:* The draw height.

```
int display_width() const noexcept;
```

4 *Returns:* The display width.

```
int display_height() const noexcept;
```

5 *Returns:* The display height.

```
vector_2d dimensions() const noexcept;
```

6 *Returns:* A `vector_2d` constructed using the draw width as the first argument and the draw height as the second argument.

```
vector_2d display_dimensions() const noexcept;
```

7 *Returns:* A `vector_2d` constructed using the display width as the first argument and the display height as the second argument.

```
experimental::io2d::scaling scaling() const noexcept;
```

8 *Returns:* The scaling type.

```
function<experimental::io2d::rectangle(const display_surface&, bool&)>
  user_scaling_callback() const;
function<experimental::io2d::rectangle(const display_surface&, bool&)>
  user_scaling_callback(error_code& ec) const noexcept;
```

9 *Returns:* A copy of user scaling callback.

10 *Throws:* As specified in Error reporting (4).

11 *Error conditions:* `errc::not_enough_memory` if a failure to allocate memory occurs.

```
optional<brush> letterbox_brush() const noexcept;
```

12 *Returns:* A `optional<brush>` object constructed using the user-provided letterbox brush or, if no user-provided letterbox brush is set, an empty `optional<brush>` object.

```
bool auto_clear() const noexcept;
```

13 *Returns:* The value of auto clear.

```
float desired_framerate() const noexcept;
```

14 *Returns:* The value of desired framerate.

```
float elapsed_draw_time() const noexcept;
```

15 *Returns:* If called from the draw callback during the execution of `display_surface::show`, the amount of time in milliseconds that has passed since the previous call to the draw callback by the current execution of `display_surface::show`; otherwise `0.0f`.

11.18 Class `mapped_surface` [io2d.mappedsurface]

11.18.1 `mapped_surface` synopsis [io2d.mappedsurface.synopsis]

```
namespace std::experimental::io2d::v1 {
  class mapped_surface {
  public:
    // 11.18.3, construct/copy/move/destroy:
    mapped_surface() = delete;
    ~mapped_surface();

    // 11.18.4, modifiers:
```

```
      void commit_changes();
      void commit_changes(error_code& ec) noexcept;
      void commit_changes(const rectangle& area);
      void commit_changes(const rectangle& area, error_code& ec) noexcept;
      unsigned char* data();
      unsigned char* data(error_code& ec) noexcept;

      // 11.18.5, observers:
      const unsigned char* data() const;
      const unsigned char* data(error_code& ec) const noexcept;
      experimental::io2d::format format() const noexcept;
      int width() const noexcept;
      int height() const noexcept;
      int stride() const noexcept;
    };
  }
```

11.18.2 `mapped_surface` description [io2d.mappedsurface.intro]

1 The `mapped_surface` class provides access to inspect and modify the pixel data of a `surface` object's underlying graphics data graphics resource or a subsection thereof.

2 A `mapped_surface` object can only be created by the `surface::map` function. Creation of a `mapped_surface` object fails if the format of the pixel data would be `format::invalid` or `format::unknown`.

3 The pixel data is presented as an array in the form of a pointer to (possibly `const`) `unsigned char`.

4 The actual format of the pixel data depends on the `format` enumerator returned by calling `mapped_surface::format` and is native-endian. For more information, see the description of the `format` enum class (11.6).

5 The pixel data array is presented as a series of horizontal rows of pixels with row `0` being the top row of pixels of the underlying graphics data graphics resource and the bottom row being the row at `mapped_surface::height() - 1`.

6 Each horizontal row of pixels begins with the leftmost pixel and proceeds right to `mapped_surface::width() - 1`.

7 The width in bytes of each horizontal row is provided by `mapped_surface::stride`. This value may be larger than the result of multiplying the width in pixels of each horizontal row by the size in bytes of the pixel's format (most commonly as a result of implementation-dependent memory alignment requirements).

8 Whether the pixel data array provides direct access to the underlying graphics data graphics resource's memory or provides indirect access as-if through a proxy or a copy is unspecified.

9 Changes made to the pixel data array are considered to be *uncommitted* so long as those changes are not reflected in the underlying graphics data graphics resource.

10 Changes made to the pixel data array are considered to be *committed* once they are reflected in the underlying graphics data graphics resource.

11.18.3 `mapped_surface` constructors and assignment operators [io2d.mappedsurface.cons]

`~mapped_surface();`

1 *Effects:* Destroys an object of type `mapped_surface`.

2 *Remarks:* Whether any uncommitted changes are committed during destruction of the `mapped_surface` object is unspecified.

3 Uncommitted changes shall not be committed during destruction of the `mapped_surface` object if doing so would result in an exception.

4 Users shall call `mapped_surface::commit_changes` to commit changes made to the surface's data prior to the destruction of the `mapped_surface` object.

11.18.4 `mapped_surface` modifiers [io2d.mappedsurface.modifiers]

```
void commit_changes();
void commit_changes(error_code& ec) noexcept;
```

1 *Effects:* Any uncommitted changes shall be committed.

2 *Throws:* As specified in Error reporting (4).

3 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
unsigned char* data();
unsigned char* data(error_code& ec) noexcept;
```

4 *Returns:* A native-endian pointer to the pixel data array. [*Example:* Given the following code:

```
image_surface imgsfc{ format::argb32, 100, 100 };
imgsfc.paint(rgba_color::red());
imgsfc.flush();
imgsfc.map([](mapped_surface& mapsfc) -> void {
    auto pixelData = mapsfc.data();
    auto p0 = static_cast<uint32_t>(pixelData[0]);
    auto p1 = static_cast<uint32_t>(pixelData[1]);
    auto p2 = static_cast<uint32_t>(pixelData[2]);
    auto p3 = static_cast<uint32_t>(pixelData[3]);
    printf("%X %X %X %X\n", p0, p1, p2, p3);
});
```

In a little-endian environment, `p0 == 0x0`, `p1 == 0x0`, `p2 == 0xFF`, and `p3 == 0xFF`.

In a big-endian environment, `p0 == 0xFF`, `p1 == 0xFF`, `p2 == 0x0`, `p3 == 0x0`. — *end example*]

5 *Remarks:* The bounds of the pixel data array range from `a`, where `a` is the address returned by this function, to `a + this->stride() * this->height()`. Given a height `h` where `h` is any value from `0` to `this->height() - 1`, any attempt to read or write a byte with an address that is not within the range of addresses defined by `a + this->stride() * h` shall result in undefined behavior; no diagnostic is required.

11.18.5 `mapped_surface` observers [io2d.mappedsurface.observers]

```
const unsigned char* data() const;
const unsigned char* data(error_code& ec) const noexcept;
```

1 *Returns:* A const native-endian pointer to the pixel data array. [*Example:* Given the following code:

```
image_surface imgsfc{ format::argb32, 100, 100 };
imgsfc.paint(rgba_color::red());
imgsfc.flush();
imgsfc.map([](mapped_surface& mapsfc) -> void {
    auto pixelData = mapsfc.data();
    auto p0 = static_cast<uint32_t>(pixelData[0]);
    auto p1 = static_cast<uint32_t>(pixelData[1]);
    auto p2 = static_cast<uint32_t>(pixelData[2]);
    auto p3 = static_cast<uint32_t>(pixelData[3]);
    printf("%X %X %X %X\n", p0, p1, p2, p3);
```

```
});
```

In a little-endian environment, `p0 == 0x0`, `p1 == 0x0`, `p2 == 0xFF`, and `p3 == 0xFF`.

In a big-endian environment, `p0 == 0xFF`, `p1 == 0xFF`, `p2 == 0x0`, `p3 == 0x0`. — *end example*]

2 *Remarks:* The bounds of the pixel data array range from `a`, where `a` is the address returned by this function, to `a + this->stride() * this->height()`. Given a height `h` where `h` is any value from `0` to `this->height() - 1`, any attempt to read a byte with an address that is not within the range of addresses defined by `a + this->stride() * h` shall result in undefined behavior; no diagnostic is required.

```
experimental::io2d::format format() const noexcept;
```

3 *Returns:* The pixel format of the mapped surface.

4 *Remarks:* If the mapped surface is invalid, this function shall return `experimental::io2d::format::invalid`.

```
int width() const noexcept;
```

5 *Returns:* The number of pixels per horizontal line of the mapped surface.

6 *Remarks:* This function shall return the value `0` if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

```
int height() const noexcept;
```

7 *Returns:* The number of horizontal lines of pixels in the mapped surface.

8 *Remarks:* This function shall return the value `0` if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

```
int stride() const noexcept;
```

9 *Returns:* The length, in bytes, of a horizontal line of the mapped surface. [*Note:* This value is at least as large as the width in pixels of a horizontal line multiplied by the number of bytes per pixel but may be larger as a result of padding. — *end note*]

10 *Remarks:* This function shall return the value `0` if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

12 Standalone functions [io2d.standalone]

12.1 Standalone functions synopsis [io2d.standalone.synopsis]

```
namespace std::experimental::io2d::v1 {
  int format_stride_for_width(format format, int width) noexcept;
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat,
    scaling scl = scaling::letterbox,
    refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f);
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat, error_code& ec,
    scaling scl = scaling::letterbox,
    refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f) noexcept;
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, scaling scl = scaling::letterbox,
    refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f);
  display_surface make_display_surface(int preferredWidth,
    int preferredHeight, format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, ::std::error_code& ec,
    scaling scl = scaling::letterbox,
    refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f) noexcept;
  image_surface make_image_surface(format format, int width, int height);
  image_surface make_image_surface(format format, int width, int height,
    error_code& ec) noexcept;
  image_surface make_image_surface(image_surface& sfc) noexcept;
  float angle_for_point(const vector_2d& ctr, const vector_2d& pt) noexcept;
  vector_2d point_for_angle(float ang, float rad = 1.0f) noexcept;
  vector_2d point_for_angle(float ang, const vector_2d& rad) noexcept;
  vector_2d arc_start(const vector_2d& ctr, float sang, const vector_2d& rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
  vector_2d arc_center(const vector_2d& cpt, float sang, const vector_2d& rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
  vector_2d arc_end(const vector_2d& cpt, float eang, const vector_2d& rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
}
```

12.2 `format_stride_for_width` [io2d.standalone.formatstrideforwidth]

```
int format_stride_for_width(format fmt, int width) noexcept;
```

1 *Returns:* The size in bytes of a row of pixels with a visual data format of `fmt` that is `width` pixels wide. This value may be larger than the value obtained by multiplying the number of bytes specified by the `format` enumerator specified by `fmt` by the number of pixels specified by `width`.

2 If `fmt == format::invalid`, this function shall return `0`.

12.3 `make_display_surface` [io2d.standalone.makedisplaysurface]

```
display_surface make_display_surface(int preferredWidth,
  int preferredHeight, format preferredFormat,
  scaling scl = scaling::letterbox,
  refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f);
```

```
display_surface make_display_surface(int preferredWidth,
  int preferredHeight, format preferredFormat, error_code& ec,
  scaling scl = scaling::letterbox,
  refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f)
  noexcept;
display_surface make_display_surface(int preferredWidth,
  int preferredHeight, format preferredFormat, int preferredDisplayWidth,
  int preferredDisplayHeight, scaling scl = scaling::letterbox,
  refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f);
display_surface make_display_surface(int preferredWidth,
  int preferredHeight, format preferredFormat, int preferredDisplayWidth,
  int preferredDisplayHeight, ::std::error_code& ec,
  scaling scl = scaling::letterbox,
  refresh_rate rr = refresh_rate::as_fast_as_possible, float fps = 30.0f)
  noexcept;
```

1 *Returns:* Returns a `display_surface` object that is exactly the same as-if the equivalent `display_surface` constructor was called with the same arguments.

2 *Throws:* As specified in Error reporting (4).

3 *Error conditions:* The errors, if any, produced by this function are the same as the errors for the equivalent `display_surface` constructor (11.17.5).

12.4 make_image_surface [io2d.standalone.makeimagesurface]

```
image_surface make_image_surface(int width, int height,
  format fmt = format::argb32);
image_surface make_image_surface(int width, int height,
  error_code& ec, format fmt = format::argb32) noexcept;
```

1 *Returns:* Returns an `image_surface` object that is exactly the same as-if the `image_surface` constructor was called with the same arguments.

2 *Throws:* As specified in Error reporting (4).

3 *Error conditions:* The errors, if any, produced by this function are the same as the errors for the equivalent `display_surface` constructor (11.16.3).

```
image_surface make_image_surface(image_surface& sfc) noexcept;
```

4 *Returns:* An exact copy of `sfc`.

5 [*Note:* The `image_surface` class intentionally does not provide copy semantics because with many modern graphics technologies, making such a copy is almost always a very time consuming operation. This function allows users to make a copy of an `image_surface` object while preserving the move-only semantics of the `image_surface` class. — *end note*]

12.5 angle_for_point [io2d.standalone.angleforpoint]

```
float angle_for_point(const vector_2d& ctr, const vector_2d& pt,
  const vector_2d& scl = vector_2d{ 1.0f, 1.0f }) noexcept;
```

1 *Returns:* The angle, in radians, of `pt` as a point on a circle with a center at `ctr`. If the angle is less that `pi<float> / 180000.0f`, returns `0.0f`.

12.6 point_for_angle [io2d.standalone.pointforangle]

```
vector_2d point_for_angle(float ang, float rad = 1.0f) noexcept;
vector_2d point_for_angle(float ang, const vector_2d& rad) noexcept;
```

1 *Requires:* If it is a `float`, `rad` is greater than `0.0f`. If it is a `vector_2d`, `rad.x()` or `rad.y()` is greater than `0.0f` and neither is less than `0.0f`.

2 *Returns:* The result of rotating the point `vector_2d{ 1.0f, 0.0f }`, around an origin of `vector_2d{ 0.0f, 0.0f }` by `ang` radians, with a positive value of `ang` meaning counterclockwise rotation and a negative value meaning clockwise rotation, with the result being multiplied by `rad`.

12.7 arc_start [io2d.standalone.arcstart]

```
vector_2d arc_start(const vector_2d& ctr, float sang, const vector_2d& rad,
  const matrix_2d& m = matrix_2d{}) noexcept;
```

1 *Requires:* `rad.x()` and `rad.y()` are both greater than `0.0f`.

2 *Returns:* As-if:

```
auto lmtx = m;
lmtx.m20(0.0f); lmtx.m21(0.0f);
auto pt = point_for_angle(sang, rad);
return ctr + pt * lmtx;
```

3 [*Note:* Among other things, this function is useful for determining the point at which a new path should begin if the first item in the path is an arc and the user wishes to clearly define its center. — *end note*]

12.8 arc_center [io2d.standalone.arccenter]

```
vector_2d arc_center(const vector_2d& cpt, float sang, const vector_2d& rad,
  const matrix_2d& m = matrix_2d{}) noexcept;
```

1 *Requires:* `rad.x()` and `rad.y()` are both greater than `0.0f`.

2 *Returns:* As-if:

```
auto lmtx = m;
lmtx.m20(0.0f); lmtx.m21(0.0f);
auto centerOffset = point_for_angle(two_pi<float> - sang, rad);
centerOffset.y(-centerOffset.y());
return cpt - centerOffset * lmtx;
```

12.9 arc_end [io2d.standalone.arcend]

```
vector_2d arc_end(const vector_2d& cpt, float eang, const vector_2d& rad,
  const matrix_2d& m = matrix_2d{}) noexcept;
```

1 *Requires:* `rad.x()` and `rad.y()` are both greater than `0.0f`.

2 *Returns:* As-if:

```
auto lmtx = m;
auto tfrm = matrix_2d::init_rotate(eang);
lmtx.m20(0.0f); lmtx.m21(0.0f);
auto pt = (rad * tfrm);
pt.y(-pt.y());
return cpt + pt * lmtx;
```

Annex A (informative) Bibliography [bibliography]

1 The following is a list of informative resources intended to assist in the understanding or use of this Technical Specification.

(1.1) — Porter, Thomas and Duff, Tom, 1984, Compositing digital images. ACM SIGGRAPH Computer Graphics. 1984. Vol. 18, no. 3, p. 253-259. DOI 10.1145/964965.808606. Association for Computing Machinery (ACM)

(1.2) — Foley, James D. et al., *Computer graphics: principles and practice.* 2nd ed. Reading, Massachusetts : Addison-Wesley, 1996.

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.