Document Number: P0267R4
Date: 2017-03-20
Revises: P0267R3

Reply to: Michael B. McLaughlin
mikebmcl@gmail.com

Herb Sutter
Microsoft Inc.
hsutter@microsoft.com

Jason Zink
jzink_1@yahoo.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

Contents ii

List of Tables vi

List of Figures vii

1 Scope 1

2 Normative references 2

3 Terms and definitions 3

3.1 standard coordinate space 3
3.2 visual data 3
3.3 channel 3
3.4 visual data format 3
3.5 visual data element 3
3.6 alpha 3
3.7 pixel 3
3.8 aliasing 3
3.9 artifact 3
3.10 anti-aliasing 3
3.11 aspect ratio 4
3.12 color model 4
3.13 additive color 4
3.14 RGB color model 4
3.15 RGBA color model 4
3.16 color space 4
3.17 sRGB color space 4
3.18 Bézier curve 4
3.19 Bézier curve 4
3.20 filter 4
3.21 graphics data 4
3.22 graphics data 5
3.23 graphics resource 5
3.24 graphics resource 5
3.25 pixmap 5
3.26 point 5
3.27 point 5
3.28 premultiplied format 5
3.29 graphics state data 5
3.30 graphics subsystem 5
3.31 normalize 6
3.32 render 6
3.33 rendering operation 6
3.34 compose 6
3.35 composing operation 6

3.36 composition algorithm . . . 6
3.37 rendering and composing operation . . . 6
3.38 sample . . . 6
3.39 color stop . . . 6
3.40 path segment . . . 6
3.41 control point . . . 6
3.42 degenerate path segment . . . 6
3.43 initial path segment . . . 7
3.44 final path segment . . . 7
3.45 path instruction . . . 7
3.46 path item . . . 7
3.47 path . . . 7
3.48 current point . . . 7
3.49 last-move-to point . . . 7
3.50 path group . . . 7
3.51 closed path . . . 7
3.52 open path . . . 7
3.53 degenerate path . . . 8

4 Requirements **9**
4.1 Namespaces and headers . . . 9
4.2 Feature test macros . . . 9
4.3 Native handles . . . 9
4.4 IEC 559 floating point support . . . 9

5 Error reporting **10**

6 Header `<experimental/io2d>` synopsis **12**

7 Error codes **18**
7.1 Enum class `io2d_error` . . . 18
7.2 Class `io2d_error_category` . . . 20

8 Colors **22**
8.1 Introduction to color . . . 22
8.2 Color usage requirements . . . 22
8.3 Class `bgra_color` . . . 22
8.4 `literals` namespace . . . 39

9 Geometry **40**
9.1 Class `vector_2d` . . . 40
9.2 Class `rectangle` . . . 42
9.3 Class `circle` . . . 44
9.4 Class `ellipse` . . . 46
9.5 Class `matrix_2d` . . . 47

10 Paths **53**
10.1 Processing paths . . . 53
10.2 Class `abs_cubic_curve` . . . 63
10.3 Class `abs_ellipse` . . . 64
10.4 Class `abs_line` . . . 66
10.5 Class `abs_move` . . . 66

10.6 Class abs_quadratic_curve . . . 67
10.7 Class abs_rectangle . . . 68
10.8 Class arc_clockwise . . . 71
10.9 Class arc_counterclockwise . . . 73
10.10 Class change_matrix . . . 75
10.11 Class change_origin . . . 75
10.12 Class close_path . . . 76
10.13 Class new_path . . . 77
10.14 Class rel_cubic_curve . . . 77
10.15 Class rel_ellipse . . . 79
10.16 Class rel_line . . . 80
10.17 Class rel_move . . . 81
10.18 Class rel_quadratic_curve . . . 82
10.19 Class rel_rectangle . . . 83
10.20 Class path_group . . . 85
10.21 Class path_builder . . . 86

11 Brushes **96**
11.1 Overview of brushes . . . 96
11.2 Gradient brushes . . . 96
11.3 Enum class wrap_mode . . . 100
11.4 Enum class filter . . . 101
11.5 Enum class brush_type . . . 102
11.6 Color stops . . . 103
11.7 Class brush . . . 104

12 Surfaces **107**
12.1 Class surface_props . . . 107
12.2 Class stroke_props . . . 108
12.3 Class brush_props . . . 110
12.4 Class mask_props . . . 112
12.5 Class clip_props . . . 114
12.6 Enum class antialias . . . 115
12.7 Enum class fill_rule . . . 116
12.8 Enum class line_cap . . . 117
12.9 Enum class line_join . . . 118
12.10 Enum class compositing_op . . . 118
12.11 Enum class format . . . 124
12.12 Enum class scaling . . . 126
12.13 Enum class refresh_rate . . . 129
12.14 Enum class image_file_format . . . 131
12.15 Class device . . . 132
12.16 Class surface . . . 133
12.17 Class image_surface . . . 142
12.18 Class display_surface . . . 144
12.19 Class mapped_surface . . . 161

13 Standalone functions **165**
13.1 Standalone functions synopsis . . . 165
13.2 format_stride_for_width . . . 165
13.3 make_display_surface . . . 165

13.4 make_image_surface 166

A Bibliography 167

Index 168

Index of library names 169

Index of implementation-defined behavior 176

List of Tables

1 io2d_error enumerator meanings 18

2 path group processing state data 53

3 wrap_mode enumerator meanings 101
4 filter enumerator meanings 102
5 brush_type enumerator meanings 102

6 antialias enumerator meanings 116
7 fill_rule enumerator meanings 117
8 line_cap enumerator meanings 117
9 line_join enumerator meanings 118
10 compositing_op basic enumerator meanings 120
11 compositing_op blend enumerator meanings 121
12 compositing_op hsl enumerator meanings 124
13 format enumerator meanings 125
14 scaling enumerator meanings 127
15 refresh_rate value meanings 130
16 imagefileformat enumerator meanings 132
17 surface rendering and composing operations 135
18 surface rendering and composing common state data 135
19 surface rendering and composing specific state data 136
20 Point transformations 136
21 Display surface observable state 146

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

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

(1.2) — ISO/IEC 2382 (all parts), *Information technology âĂŤ 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/IEC 15948 *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG) Functional specification*

(1.6) — 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*

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

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

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

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

(1.11) — 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 datastream and associated file format described in ISO/IEC 15948 is hereinafter called the *PNG format*.

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

5 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]

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

3.1 [io2d.defns.standardcoordinatespace]

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 clockwise

3.2 [io2d.defns.visualdata]

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

3.3 [io2d.defs.channel]

channel
a bounded set of homogeneously-spaced real numbers in the range $[0, 1]$

3.4 [io2d.defns.visualdataformat]

visual data format
a specification of visual data channels which defines a total bit size for the format and each channel's role, bit size, and location relative to the upper (high-order) bit [*Note:* The total bit size may be larger than the sum of the bit sizes of all of the channels of the format. — *end note*]

3.5 [io2d.defns.visualdataelement]

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.colormodel]

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

3.13 [io2d.defns.additivecolor]

additive color
a color defined by the emissive intensity of its color channels

3.14 [io2d.defns.rgbcolormodel]

RGB color model
an additive color model using red, green, and blue color channels

3.15 [io2d.defns.rgbacolormodel]

RGBA color model
the RGB color model with an alpha channel

3.16 [io2d.defns.colorspace]

color space
an unambiguous mapping of values to colorimetric colors

3.17 [io2d.defns.srgbcolorspace]

sRGB color space
an 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> a curve defined by the equation $f(t) = (1-t)^3 \times P_0 + 3 \times t \times (1-t)^2 \times P_1 + 3 \times t^2 \times (1-t) \times P_2 + t^3 \times P_3$ where $0.0 \leq t \leq 1.0$, 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> a curve defined by the equation $f(t) = (1-t)^2 \times P_0 + 2 \times t \times (1-t) \times P_1 + t^2 \times (1-t) \times P_2$ where $0.0 \leq t \leq 1.0$, P_0 is the starting point, P_1 is the control point, and P_2 is end point

3.20 [io2d.defns.filter]

filter
a 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> an 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> an object of unspecified type used by an implementation to provide access to and allow manipulation of visual data

3.25 [io2d.defns.pixmap]

pixmap

a raster graphics data graphics resource

3.26 [io2d.defns.point]

point

<point> a 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> a 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

a 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 {1.0, 0.0, 0.0, 0.5}. As such, in premultiplied, normalized format it would become {0.5, 0.0, 0.0, 0.5} 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.pathsegment]

path segment

a line, Bézier curve, or arc, each of which has a start point and an end point

3.41 [io2d.defns.controlpoint]

control point

a point other than the start point and end point that is used in defining a Bézier curve

3.42 [io2d.defns.degeneratepathsegment]

degenerate path segment

a path segment that has the same values for its start point, end point, and, if any, control points

3.43 [io2d.defns.initialpathsegment]

initial path segment

a 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.finalpathsegment]

final path segment

a path segment whose end point shall not be used to 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.pathinstruction]

path instruction

an instruction that creates a new path, closes an existing path, adds a geometry as a new closed path, or modifies the interpretation of path segments that follow it

3.46 [io2d.defns.pathitem]

path item

a path segment or path instruction

3.47 [io2d.defns.path]

path

a 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

3.48 [io2d.defns.currentpoint]

current point

a point established by various operations used in creating a path [*Note:* A new path has no current point except as otherwise specified. — *end note*]

3.49 [io2d.defns.lastmovetopoint]

last-move-to point

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

3.50 [io2d.defns.pathgroup]

path group

a collection of paths

3.51 [io2d.defns.closedpath]

closed path

a path with one or more path segments where the last-move-to point is used to define the end point of the path's final path segment

3.52 [io2d.defns.openpath]

open path

a path with one or more path segments where the last-move-to 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*]

3.53 [io2d.defns.degeneratepath]

degenerate path

a path with only one path segment [*Note:* The path segment is not required to be a degenerate path segment. — *end note*]

4 Requirements [io2d.req]

4.1 Namespaces and headers [io2d.req.namespace]

1 The components described in this technical specification are experimental and not part of the C++ standard library. All components described in this technical specification are declared in namespace `std::experimental::io2d::v1` or a sub-namespace thereof unless otherwise specified. The header described in this technical specification shall import the contents of `std::experimental::io2d::v1` into `std::experimental::io2d` as-if by

2

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

3 Unless otherwise specified, references to other entities described in this Technical Specification are assumed to be qualified with `std::experimental::io2d::v1::`, and references to entities described in the C++ standard are assumed to be qualified with `std::`.

4.2 Feature test macros [io2d.req.macros]

1 This macro allows users to determine which version of this Technical Specification is supported by header `<experimental/io2d>`.

2 Header `<experimental/io2d>` shall supply the following macro definition:

3 `#define __cpp_lib_experimental_io2d 201707`

4 [*Note:* The value of macro __cpp_lib_experimental_io2d is yyyymm where yyyy is the year and mm the month when the version of the Technical Specification was completed. — *end note*]

4.3 Native handles [io2d.req.native]

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

4.4 IEC 559 floating point support [io2d.req.iec559]

1 Certain requirements of this Technical Specification assume that `numeric_limits<double>::is_iec559` evaluates to `true`.

2 The behavior of these requirements is implementation-defined if `numeric_limits<double>::is_iec559` evaluates to `false`.

5 Error reporting [io2d.err.report]

1 2D graphics library functions often provide two overloads, one that throws an exception to report graphics subsystem errors, and another that sets an `error_code`.

2 [*Note:* This supports two common use cases:

(2.1) — Uses where graphics subsystem errors are truly exceptional and indicate a serious failure. Throwing an exception is the most appropriate response. This is the preferred default for most everyday programming.

(2.2) — Uses where graphics subsystem errors are routine and do not necessarily represent failure. Returning an error code is the most appropriate response. This allows application specific error handling, including simply ignoring the error.

— *end note*]

3 Functions **not** having an argument of type `error_code&` report errors as follows, unless otherwise specified:

(3.1) — When a call by the implementation to an operating system or other underlying API results in an error that prevents the function from meeting its specifications and the cause of the error is described in the function's *Error conditions* description:

(3.1.1) — If the description calls for `errc::argument_out_of_domain` or `io2d_error::invalid_index`, the exception type shall be `out_of_range` constructed with an implementation-defined `what_arg` argument value.

(3.1.2) — If the description calls for `errc::invalid_argument`, the exception type shall be `invalid_argument` constructed with an implementation-defined `what_arg` argument value.

(3.1.3) — If the description calls for `errc::not_enough_memory`, the error shall be reported by throwing an exception as described in C++ 2014 §17.6.5.12 [res.on.exception.handling].

(3.1.4) — In all other cases the exception type shall be `system_error` constructed with an `ec` argument value formed by passing the specified enumerator value to `make_error_code` and an implementation-defined `what_arg` argument value, unless otherwise specified.

(3.2) — When a call by the implementation to an operating system or other underlying API results in an error that prevents the function from meeting its specifications and the cause of the error is **not** described in the function's *Error conditions* description and is not a failure to allocate storage, an exception of type `system_error` shall be thrown constructed with its `error_code` argument set as appropriate for the specific operating system dependent error. Implementations shall document the cause, enumerator value, `error_category`, and exception type for each of these additional error conditions.

(3.3) — Failure to allocate storage is reported by throwing an exception as described in C++ 2014 §17.6.5.12 [res.on.exception.handling].

(3.4) — Destructors throw nothing.

4 Functions taking an argument of type `error_code&` report errors as follows, unless otherwise specified:

(4.1) — When a call by the implementation to an operating system or other underlying API results in an error that prevents the function from meeting its specifications and the cause of the error is described in the function's *Error conditions* description, the `error_code&` argument is set as appropriate for the specified enumerator.

(4.2) — When a call by the implementation to an operating system or other underlying API results in an error that prevents the function from meeting its specifications and the cause of the error is **not** described in the function's *Error conditions* description and is not a failure to allocate storage, the `error_code&` argument is set as appropriate for the specific operating system dependent error. Implementations should document these errors where possible.

(4.3) — If a failure to allocate storage occurs, the `error_code&` argument shall be set to `make_error_code(errc::not_enough_memory)`.

(4.4) — Otherwise, `clear()` is called on the `error_code&` argument.

6 Header <experimental/io2d> synopsis [syn]

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

  struct nullvalue_t;
  constexpr nullvalue_t nullvalue{ implementation-defined };

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

  enum class io2d_error;
  enum class antialias;
  enum class content;
  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 subpixel_order;
  enum class scaling;
  enum class refresh_rate;

  class io2d_error_category;
  const error_category& io2d_category() noexcept;

  class rectangle;
  class circle;

  class bgra_color;

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

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

  class matrix_2d;
```

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

namespace path_data {
  class abs_cubic_curve;
  class abs_ellipse;
  class abs_line;
  class abs_move;
  class abs_quadratic_curve;
  class abs_rectangle;
  class arc_clockwise;
  class arc_counterclockwise;
  class change_matrix;
  class change_origin;
  class close_path;
  class new_path;
  class rel_cubic_curve;
  class rel_ellipse;
  class rel_line;
  class rel_move;
  class rel_quadratic_curve;
  class rel_rectangle;
  using path_data_types = typename variant<abs_cubic_curve, abs_ellipse,
    abs_line, abs_move, abs_quadratic_curve, abs_rectangle, arc_clockwise,
    arc_counterclockwise, change_matrix, change_origin, close_path, new_path,
    rel_cubic_curve, rel_ellipse, rel_line, rel_move, rel_quadratic_curve, rel_rectangle>;
};

class path;

template <class Allocator = allocator<path_data::path_data_types>>
class path_builder {
public:
  using value_type = path_data::path_data_types;
  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>;

  // 10.21.3, 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;

  // 10.21.6, 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;

  // 10.21.4, 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;

  // 10.21.5, modifiers:
  void new_path() noexcept;
  void close_path() noexcept;
  void arc_clockwise(const vector_2d& center, double radius, double angle1,
  double angle2) noexcept;
  void arc_counterclockwise(const vector_2d& center, double radius,
  double angle1, double angle2) noexcept;
  void cubic_curve_to(const vector_2d& pt0, const vector_2d& pt1,
  const vector_2d& pt2) noexcept;
  void line_to(const vector_2d& pt) noexcept;
  void move_to(const vector_2d& pt) noexcept;
  void quadratic_curve_to(const vector_2d& pt0, const vector_2d& pt2)
  noexcept;
  void rectangle(const experimental::io2d::rectangle& r) noexcept;
  void rel_cubic_curve_to(const vector_2d& dpt0, const vector_2d& dpt1,
  const vector_2d& dpt2) noexcept;
  void rel_line_to(const vector_2d& dpt) noexcept;
  void rel_move_to(const vector_2d& dpt) noexcept;
  void rel_quadratic_curve_to(const vector_2d& pt0, const vector_2d& pt2)
  noexcept;
  void transform_matrix(const matrix_2d& m) noexcept;
  void origin(const vector_2d& pt) 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;

  // 10.21.7, observers:
  experimental::io2d::rectangle path_extents() const noexcept;
  bool has_current_point() const noexcept;
  vector_2d current_point() const;
  vector_2d current_point(error_code& ec) const noexcept;
  matrix_2d transform_matrix() const noexcept;
  vector_2d origin() const 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);

    // 10.21.8, specialized algorithms:
    template <class Allocator>
    void swap(path_builder<Allocator>& lhs, path_builder<Allocator>& rhs)
      noexcept(noexcept(lhs.swap(rhs)));

    class device;

    class brush;

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

    template <class T>
    constexpr T pi = T(3.14159265358979323846264338327950288L);

    template <class T>
    constexpr T two_pi = T(6.28318530717958647692528676655900576L);

    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;
  } } } }

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

    template<>
    struct is_error_code_enum<implementation-defined>
      : public true_type{ }; // exposition only

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

    std::error_code make_error_code(experimental::io2d::io2d_error e) noexcept;
  }
```

7 Error codes [errorcodes]

1 The `io2d_error` enum class and the `io2d_error_category` class are provided by this Technical Specification to report errors from the graphics subsystem, excluding certain errors which shall be reported in other ways as per the requirements of (5).

7.1 Enum class `io2d_error` [io2derror]

7.1.1 `io2d_error` Summary [io2derror.summary]

1 The `io2d_error` enum class is an enumeration holding error condition values which are used with the `io2d_error_category` class. See Table 1 for the meaning of each error condition value.

7.1.2 `io2d_error` Synopsis [io2derror.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class io2d_error {
    success,
    invalid_pop_state,
    no_current_point,
    invalid_matrix,
    invalid_status,
    null_pointer,
    invalid_string,
    invalid_path_data,
    read_error,
    write_error,
    surface_finished,
    invalid_dash,
    invalid_index,
    clip_not_representable,
    invalid_stride,
    device_error,
    invalid_mesh_construction,
  };
} } }

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

7.1.3 `io2d_error` Enumerators [io2derror.enumerators]

Table 1 — `io2d_error` enumerator meanings

Enumerator	Meaning
`success`	The operation completed successfully.
`invalid_pop_state`	A call was made to `surface::pop_state` for which no prior call to `surface::push_state` was made.

Table 1 — `io2d_error` enumerator meanings (continued)

Enumerator	Meaning
`no_current_point`	A path segment or path instruction encountered during path processing requires a value for current point but current point has no value.
`invalid_matrix`	A `matrix_2d` value that the operation depends on is invalid. Except as otherwise specified, this means that the `matrix_2d` value is not invertible.
`invalid_status`	An internal error has occurred. The conditions and circumstances under which this `io2d_error` value occurs are implementation-defined. [*Note:* This value should only be used when no other `io2d_error` value is appropriate. It signifies that an implementation-specific error occurred such as passing a bad native handle as an argument. — *end note*]
`null_pointer`	A null pointer value was unexpectedly encountered. The conditions and circumstances under which this `io2d_error` value occurs are implementation-defined.
`invalid_string`	A UTF-8 string value was expected but the string is not a valid UTF-8 string.
`invalid_path_data`	Invalid data was encountered in a `path_group` or a `path_builder` object. [*Note:* This status value should only occur when a user creates invalid path data and appends it to a path. — *end note*]
`read_error`	An error occurred while attempting to read data from an input stream.
`write_error`	An error occurred while attempting to write data to an output stream.
`surface_finished`	An attempt was made to use or manipulate a `surface` object or `surface`-derived object which is no longer valid. [*Note:* This can occur due to a previous call to `surface::finish` or as a result of erroneous usage of a native handle. — *end note*]
`invalid_dash`	An invalid dash value was specified in a call to `surface::set_dashes`.
`invalid_index`	An index value was specified in a call to a function which is outside the range of index values that are currently valid.
`clip_not_representable`	A call was made to `surface::get_clip_rectangles` when the `surface` object's current clipping region could not be represented with rectangles.
`invalid_stride`	An invalid stride value was used. Surface formats may require padding at the end of each row of pixel data depending on the implementation and the current graphics chipset, if any. Use `format_stride_for_width` to obtain the correct stride value.
`device_error`	The operation failed. The `device` encountered an error. [*Note:* The conditions and circumstances in which this `io2d_error` value occurs are implementation-defined. — *end note*]

7.2 Class io2d_error_category [io2derrcat]

7.2.1 io2d_error_category synopsis [io2derrcat.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class io2d_error_category : public std::error_category {
  public:
    // 7.2.3, observers:
    virtual const char* name() const noexcept override;
    virtual string message(int errVal) const override;
    virtual bool equivalent(int code,
      const error_condition& condition) const noexcept override;
    virtual bool equivalent(const error_code& ec,
      int condition) const noexcept override;
  };

  // 7.2.4, non-member functions:
  const error_category& io2d_category() noexcept;
} } }

  error_condition make_error_condition(
    experimental::io2d::io2d_error e) noexcept;
  error_code make_error_code(experimental::io2d::io2d_error e) noexcept;
}
```

7.2.2 io2d_error_category Description [io2derrcat.intro]

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

7.2.3 io2d_error_category observers [io2derrcat.observers]

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

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

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

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

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

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

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

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

7.2.4 `io2d_error_category` non-member functions [io2derrcat.nonmembers]

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

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

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

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

3 *Returns:* `error_condition(static_cast<int>(e), experimental::io2d::io2d_category());`.

```
error_code make_error_code(experimental::io2d::io2d_error e) noexcept;
```

4 *Returns:* `error_code(static_cast<int>(e), experimental::io2d::io2d_category());`.

8 Colors [colors]

8.1 Introduction to color [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.

8.2 Color usage requirements [colors.reqs]

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

8.3 Class `bgra_color` [bgracolor]

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

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

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

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

5 The type predefines a set of named colors, for which each channel is an unsigned normalized 8-bit integer.

8.3.1 bgra_color synopsis [bgracolor.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class bgra_color {
    // 8.3.2, construct/copy/move/destroy:
    constexpr bgra_color() noexcept;
    constexpr bgra_color(const bgra_color&) noexcept = default;
    constexpr bgra_color& operator=(const bgra_color&) noexcept = default;
    bgra_color(bgra_color&&) noexcept = default;
    bgra_color& operator=(bgra_color&&) noexcept = default;
    constexpr bgra_color(double r, double g, double b, double a = 1.0) noexcept;

    // 8.3.3, modifiers:
    void r(double val) noexcept;
    void g(double val) noexcept;
    void b(double val) noexcept;
    void a(double val) noexcept;

    // 8.3.4, observers:
    constexpr double r() const noexcept;
    constexpr double g() const noexcept;
    constexpr double b() const noexcept;
    constexpr double a() const noexcept;

    // 8.3.5, static member functions:
    static const bgra_color& alice_blue() noexcept;
    static const bgra_color& antique_white() noexcept;
    static const bgra_color& aqua() noexcept;
    static const bgra_color& aquamarine() noexcept;
    static const bgra_color& azure() noexcept;
    static const bgra_color& beige() noexcept;
    static const bgra_color& bisque() noexcept;
    static const bgra_color& black() noexcept;
    static const bgra_color& blanched_almond() noexcept;
    static const bgra_color& blue() noexcept;
    static const bgra_color& blue_violet() noexcept;
    static const bgra_color& brown() noexcept;
    static const bgra_color& burly_wood() noexcept;
    static const bgra_color& cadet_blue() noexcept;
    static const bgra_color& chartreuse() noexcept;
    static const bgra_color& chocolate() noexcept;
    static const bgra_color& coral() noexcept;
    static const bgra_color& cornflower_blue() noexcept;
    static const bgra_color& cornsilk() noexcept;
    static const bgra_color& crimson() noexcept;
    static const bgra_color& cyan() noexcept;
    static const bgra_color& dark_blue() noexcept;
    static const bgra_color& dark_cyan() noexcept;
    static const bgra_color& dark_goldenrod() noexcept;
    static const bgra_color& dark_gray() noexcept;
    static const bgra_color& dark_green() noexcept;
    static const bgra_color& dark_grey() noexcept;
    static const bgra_color& dark_khaki() noexcept;
    static const bgra_color& dark_magenta() noexcept;
    static const bgra_color& dark_olive_green() noexcept;
    static const bgra_color& dark_orange() noexcept;
    static const bgra_color& dark_orchid() noexcept;
    static const bgra_color& dark_red() noexcept;
    static const bgra_color& dark_salmon() noexcept;
    static const bgra_color& dark_sea_green() noexcept;
    static const bgra_color& dark_slate_blue() noexcept;
    static const bgra_color& dark_slate_gray() noexcept;
    static const bgra_color& dark_slate_grey() noexcept;
    static const bgra_color& dark_turquoise() noexcept;
    static const bgra_color& dark_violet() noexcept;
    static const bgra_color& deep_pink() noexcept;
    static const bgra_color& deep_sky_blue() noexcept;
    static const bgra_color& dim_gray() noexcept;
    static const bgra_color& dim_grey() noexcept;
    static const bgra_color& dodger_blue() noexcept;
    static const bgra_color& firebrick() noexcept;
    static const bgra_color& floral_white() noexcept;
    static const bgra_color& forest_green() noexcept;
    static const bgra_color& fuchsia() noexcept;
    static const bgra_color& gainsboro() noexcept;
    static const bgra_color& ghost_white() noexcept;
    static const bgra_color& gold() noexcept;
    static const bgra_color& goldenrod() noexcept;
    static const bgra_color& gray() noexcept;
    static const bgra_color& green() noexcept;
    static const bgra_color& green_yellow() noexcept;
    static const bgra_color& grey() noexcept;
    static const bgra_color& honeydew() noexcept;
    static const bgra_color& hot_pink() noexcept;
    static const bgra_color& indian_red() noexcept;
    static const bgra_color& indigo() noexcept;
    static const bgra_color& ivory() noexcept;
    static const bgra_color& khaki() noexcept;
    static const bgra_color& lavender() noexcept;
    static const bgra_color& lavender_blush() noexcept;
    static const bgra_color& lawn_green() noexcept;
    static const bgra_color& lemon_chiffon() noexcept;
    static const bgra_color& light_blue() noexcept;
    static const bgra_color& light_coral() noexcept;
    static const bgra_color& light_cyan() noexcept;
    static const bgra_color& light_goldenrod_yellow() noexcept;
    static const bgra_color& light_gray() noexcept;
    static const bgra_color& light_green() noexcept;
    static const bgra_color& light_grey() noexcept;
    static const bgra_color& light_pink() noexcept;
    static const bgra_color& light_salmon() noexcept;
    static const bgra_color& light_sea_green() noexcept;
    static const bgra_color& light_sky_blue() noexcept;
    static const bgra_color& light_slate_gray() noexcept;
    static const bgra_color& light_slate_grey() noexcept;
    static const bgra_color& light_steel_blue() noexcept;
    static const bgra_color& light_yellow() noexcept;
    static const bgra_color& lime() noexcept;
    static const bgra_color& lime_green() noexcept;
    static const bgra_color& linen() noexcept;
    static const bgra_color& magenta() noexcept;
    static const bgra_color& maroon() noexcept;
    static const bgra_color& medium_aquamarine() noexcept;
    static const bgra_color& medium_blue() noexcept;
    static const bgra_color& medium_orchid() noexcept;
    static const bgra_color& medium_purple() noexcept;
    static const bgra_color& medium_sea_green() noexcept;
    static const bgra_color& medium_slate_blue() noexcept;
    static const bgra_color& medium_spring_green() noexcept;
    static const bgra_color& medium_turquoise() noexcept;
    static const bgra_color& medium_violet_red() noexcept;
    static const bgra_color& midnight_blue() noexcept;
    static const bgra_color& mint_cream() noexcept;
    static const bgra_color& misty_rose() noexcept;
    static const bgra_color& moccasin() noexcept;
    static const bgra_color& navajo_white() noexcept;
    static const bgra_color& navy() noexcept;
    static const bgra_color& old_lace() noexcept;
    static const bgra_color& olive() noexcept;
    static const bgra_color& olive_drab() noexcept;
    static const bgra_color& orange() noexcept;
    static const bgra_color& orange_red() noexcept;
    static const bgra_color& orchid() noexcept;
    static const bgra_color& pale_goldenrod() noexcept;
    static const bgra_color& pale_green() noexcept;
    static const bgra_color& pale_turquoise() noexcept;
    static const bgra_color& pale_violet_red() noexcept;
    static const bgra_color& papaya_whip() noexcept;
    static const bgra_color& peach_puff() noexcept;
    static const bgra_color& peru() noexcept;
    static const bgra_color& pink() noexcept;
    static const bgra_color& plum() noexcept;
    static const bgra_color& powder_blue() noexcept;
    static const bgra_color& purple() noexcept;
    static const bgra_color& red() noexcept;
    static const bgra_color& rosy_brown() noexcept;
    static const bgra_color& royal_blue() noexcept;
    static const bgra_color& saddle_brown() noexcept;
    static const bgra_color& salmon() noexcept;
    static const bgra_color& sandy_brown() noexcept;
    static const bgra_color& sea_green() noexcept;
    static const bgra_color& sea_shell() noexcept;
    static const bgra_color& sienna() noexcept;
    static const bgra_color& silver() noexcept;
    static const bgra_color& sky_blue() noexcept;
    static const bgra_color& slate_blue() noexcept;
    static const bgra_color& slate_gray() noexcept;
    static const bgra_color& slate_grey() noexcept;
    static const bgra_color& snow() noexcept;
    static const bgra_color& spring_green() noexcept;
    static const bgra_color& steel_blue() noexcept;
    static const bgra_color& tan() noexcept;
    static const bgra_color& teal() noexcept;
    static const bgra_color& thistle() noexcept;
    static const bgra_color& tomato() noexcept;
    static const bgra_color& transparent_black() noexcept;
```

```
      static const bgra_color& turquoise() noexcept;
      static const bgra_color& violet() noexcept;
      static const bgra_color& wheat() noexcept;
      static const bgra_color& white() noexcept;
      static const bgra_color& white_smoke() noexcept;
      static const bgra_color& yellow() noexcept;
      static const bgra_color& yellow_green() noexcept;
    };

    // 8.3.6, non-member operators:
    bool operator==(const bgra_color& lhs, const bgra_color& rhs) noexcept;
    bool operator!=(const bgra_color& lhs, const bgra_color& rhs) noexcept;
  } } } }
```

8.3.2 bgra_color constructors and assignment operators [bgracolor.cons]

`constexpr bgra_color(double r, double g, double b, double a = 1.0) noexcept;`

1 *Requires:* `r >= 0.0` and `r <= 1.0` and `g >= 0.0` and `g <= 1.0` and `b >= 0.0` and `b <= 1.0`. Where there is an `a` parameter, `a >= 0.0` and `a <= 1.0`.

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

3 The alpha channel shall be set to the value of `a`.

4 The red channel shall be set to `r` multiplied by the value of `a`.

5 The green channel shall be set to `g` multiplied by the value of `a`.

6 The blue channel shall be set to `b` multiplied by the value of `a`.

8.3.3 bgra_color modifiers [bgracolor.modifiers]

`void r(double val) noexcept;`

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

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

`void g(double val) noexcept;`

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

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

`void b(double val) noexcept;`

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

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

`void a(double val) noexcept;`

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

8 *Effects:* The alpha channel shall be set to `val`.

8.3.4 bgra_color observers [bgracolor.observers]

`constexpr double r() const noexcept;`

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

`constexpr double g() const noexcept;`

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

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

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

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

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

8.3.5 `bgra_color` static member functions [bgracolor.statics]

```
static const bgra_color& alice_blue() noexcept;
```

1 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 240ubyte, 248ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& antique_white() noexcept;
```

2 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 250ubyte, 235ubyte, 215ubyte, 255ubyte }`.

```
static const bgra_color& aqua() noexcept;
```

3 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 255ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& aquamarine() noexcept;
```

4 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 127ubyte, 255ubyte, 212ubyte, 255ubyte }`.

```
static const bgra_color& azure() noexcept;
```

5 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 240ubyte, 255ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& beige() noexcept;
```

6 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 245ubyte, 245ubyte, 220ubyte, 255ubyte }`.

```
static const bgra_color& bisque() noexcept;
```

7 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 228ubyte, 196ubyte, 255ubyte }`.

```
static const bgra_color& black() noexcept;
```

8 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 0ubyte, 0ubyte, 255ubyte }`.

```
static const bgra_color& blanched_almond() noexcept;
```

9 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 235ubyte, 205ubyte, 255ubyte }`.

```
static const bgra_color& blue() noexcept;
```

10 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 0ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& blue_violet() noexcept;
```

11 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 138ubyte, 43ubyte, 226ubyte, 255ubyte }`.

```
static const bgra_color& brown() noexcept;
```

12 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 165ubyte, 42ubyte, 42ubyte, 255ubyte }`.

```
static const bgra_color& burly_wood() noexcept;
```

13 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 222ubyte, 184ubyte, 135ubyte, 255ubyte }`.

```
static const bgra_color& cadet_blue() noexcept;
```

14 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 95ubyte, 158ubyte, 160ubyte, 255ubyte }`.

```
static const bgra_color& chartreuse() noexcept;
```

15 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 127ubyte, 255ubyte, 0ubyte, 255ubyte }`.

```
static const bgra_color& chocolate() noexcept;
```

16 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 210ubyte, 105ubyte, 30ubyte, 255ubyte }`.

```
static const bgra_color& coral() noexcept;
```

17 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 127ubyte, 80ubyte, 255ubyte }`.

```
static const bgra_color& cornflower_blue() noexcept;
```

18 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 100ubyte, 149ubyte, 237ubyte, 255ubyte }`.

```
static const bgra_color& cornsilk() noexcept;
```

19 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 248ubyte, 220ubyte, 255ubyte }`.

```
static const bgra_color& crimson() noexcept;
```

20 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 220ubyte, 20ubyte, 60ubyte, 255ubyte }`.

```
static const bgra_color& cyan() noexcept;
```

21 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 255ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& dark_blue() noexcept;
```

22 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 0ubyte, 139ubyte, 255ubyte }`.

```
static const bgra_color& dark_cyan() noexcept;
```

23 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 139ubyte, 139ubyte, 255ubyte }`.

```
static const bgra_color& dark_goldenrod() noexcept;
```

24 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 184ubyte, 134ubyte, 11ubyte, 255ubyte }`.

```
static const bgra_color& dark_gray() noexcept;
```

25 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 169ubyte, 169ubyte, 169ubyte, 255ubyte }`.

```
static const bgra_color& dark_green() noexcept;
```

26 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 100ubyte, 0ubyte, 255ubyte }`.

```
static const bgra_color& dark_grey() noexcept;
```

27 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 169ubyte, 169ubyte, 169ubyte, 255ubyte }`.

```
static const bgra_color& dark_khaki() noexcept;
```

28 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 189ubyte, 183ubyte, 107ubyte, 255ubyte }`.

```
static const bgra_color& dark_magenta() noexcept;
```

29 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 139ubyte, 0ubyte, 139ubyte, 255ubyte }`.

```
static const bgra_color& dark_olive_green() noexcept;
```

30 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 85ubyte, 107ubyte, 47ubyte, 255ubyte }`.

```
static const bgra_color& dark_orange() noexcept;
```

31 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 140ubyte, 0ubyte, 255ubyte }`.

```
static const bgra_color& dark_orchid() noexcept;
```

32 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 153ubyte, 50ubyte, 204ubyte, 255ubyte }`.

```
static const bgra_color& dark_red() noexcept;
```

33 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 139ubyte, 0ubyte, 0ubyte, 255ubyte }`.

```
static const bgra_color& dark_salmon() noexcept;
```

34 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 233ubyte, 150ubyte, 122ubyte, 255ubyte }`.

```
static const bgra_color& dark_sea_green() noexcept;
```

35 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 143ubyte, 188ubyte, 143ubyte, 255ubyte }`.

```
static const bgra_color& dark_slate_blue() noexcept;
```

36 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 72ubyte, 61ubyte, 139ubyte, 255ubyte }`.

```
static const bgra_color& dark_slate_gray() noexcept;
```

37 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 47ubyte, 79ubyte, 79ubyte, 255ubyte }`.

```
static const bgra_color& dark_slate_grey() noexcept;
```

38 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 47ubyte, 79ubyte, 79ubyte, 255ubyte }`.

```
static const bgra_color& dark_turquoise() noexcept;
```

39 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 206ubyte, 209ubyte, 255ubyte }`.

```
static const bgra_color& dark_violet() noexcept;
```

40 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 148ubyte, 0ubyte, 211ubyte, 255ubyte }`.

```
static const bgra_color& deep_pink() noexcept;
```

41 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 20ubyte, 147ubyte, 255ubyte }`.

```
static const bgra_color& deep_sky_blue() noexcept;
```

42 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 191ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& dim_gray() noexcept;
```

43 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 105ubyte, 105ubyte, 105ubyte, 255ubyte }`.

```
static const bgra_color& dim_grey() noexcept;
```

44 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 105ubyte, 105ubyte, 105ubyte, 255ubyte }`.

```
static const bgra_color& dodger_blue() noexcept;
```

45 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 30ubyte, 144ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& firebrick() noexcept;
```

46 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 178ubyte, 34ubyte, 34ubyte, 255ubyte }`.

`static const bgra_color& floral_white() noexcept;`

47 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 250ubyte, 240ubyte, 255ubyte }`.

`static const bgra_color& forest_green() noexcept;`

48 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 34ubyte, 139ubyte, 34ubyte, 255ubyte }`.

`static const bgra_color& fuchsia() noexcept;`

49 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 0ubyte, 255ubyte, 255ubyte }`.

`static const bgra_color& gainsboro() noexcept;`

50 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 220ubyte, 220ubyte, 220ubyte, 255ubyte }`.

`static const bgra_color& ghost_white() noexcept;`

51 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 248ubyte, 248ubyte, 255ubyte, 255ubyte }`.

`static const bgra_color& gold() noexcept;`

52 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 215ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& goldenrod() noexcept;`

53 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 218ubyte, 165ubyte, 32ubyte, 255ubyte }`.

`static const bgra_color& gray() noexcept;`

54 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 128ubyte, 128ubyte, 128ubyte, 255ubyte }`.

`static const bgra_color& green() noexcept;`

55 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 128ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& green_yellow() noexcept;`

56 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 173ubyte, 255ubyte, 47ubyte, 255ubyte }`.

`static const bgra_color& grey() noexcept;`

57 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 128ubyte, 128ubyte, 128ubyte, 255ubyte }`.

`static const bgra_color& honeydew() noexcept;`

58 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 240ubyte, 255ubyte, 240ubyte, 255ubyte }`.

`static const bgra_color& hot_pink() noexcept;`

59 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 105ubyte, 180ubyte, 255ubyte }`.

`static const bgra_color& indian_red() noexcept;`

60 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 205ubyte, 92ubyte, 92ubyte, 255ubyte }`.

`static const bgra_color& indigo() noexcept;`

61 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 75ubyte, 0ubyte, 130ubyte, 255ubyte }`.

`static const bgra_color& ivory() noexcept;`

62 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 255ubyte, 240ubyte, 255ubyte }`.

`static const bgra_color& khaki() noexcept;`

63 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 240ubyte, 230ubyte, 140ubyte, 255ubyte }`.

`static const bgra_color& lavender() noexcept;`

64 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 230ubyte, 230ubyte, 250ubyte, 255ubyte }`.

`static const bgra_color& lavender_blush() noexcept;`

65 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 240ubyte, 245ubyte, 255ubyte }`.

`static const bgra_color& lawn_green() noexcept;`

66 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 124ubyte, 252ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& lemon_chiffon() noexcept;`

67 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 250ubyte, 205ubyte, 255ubyte }`.

`static const bgra_color& light_blue() noexcept;`

68 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 173ubyte, 216ubyte, 230ubyte, 255ubyte }`.

`static const bgra_color& light_coral() noexcept;`

69 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 240ubyte, 128ubyte, 128ubyte, 255ubyte }`.

`static const bgra_color& light_cyan() noexcept;`

70 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 224ubyte, 255ubyte, 255ubyte, 255ubyte }`.

```
static const bgra_color& light_goldenrod_yellow() noexcept;
```

71 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 250ubyte, 250ubyte, 210ubyte, 255ubyte }`.

```
static const bgra_color& light_gray() noexcept;
```

72 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 211ubyte, 211ubyte, 211ubyte, 255ubyte }`.

```
static const bgra_color& light_green() noexcept;
```

73 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 144ubyte, 238ubyte, 144ubyte, 255ubyte }`.

```
static const bgra_color& light_grey() noexcept;
```

74 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 211ubyte, 211ubyte, 211ubyte, 255ubyte }`.

```
static const bgra_color& light_pink() noexcept;
```

75 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 182ubyte, 193ubyte, 255ubyte }`.

```
static const bgra_color& light_salmon() noexcept;
```

76 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 160ubyte, 122ubyte, 255ubyte }`.

```
static const bgra_color& light_sea_green() noexcept;
```

77 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 32ubyte, 178ubyte, 170ubyte, 255ubyte }`.

```
static const bgra_color& light_sky_blue() noexcept;
```

78 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 135ubyte, 206ubyte, 250ubyte, 255ubyte }`.

```
static const bgra_color& light_slate_gray() noexcept;
```

79 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 119ubyte, 136ubyte, 153ubyte, 255ubyte }`.

```
static const bgra_color& light_slate_grey() noexcept;
```

80 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 119ubyte, 136ubyte, 153ubyte, 255ubyte }`.

```
static const bgra_color& light_steel_blue() noexcept;
```

81 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 176ubyte, 196ubyte, 222ubyte, 255ubyte }`.

```
static const bgra_color& light_yellow() noexcept;
```

82 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 255ubyte, 224ubyte, 255ubyte }`.

`static const bgra_color& lime() noexcept;`

83 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 255ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& lime_green() noexcept;`

84 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 50ubyte, 205ubyte, 50ubyte, 255ubyte }`.

`static const bgra_color& linen() noexcept;`

85 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 250ubyte, 240ubyte, 230ubyte, 255ubyte }`.

`static const bgra_color& magenta() noexcept;`

86 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 0ubyte, 255ubyte, 255ubyte }`.

`static const bgra_color& maroon() noexcept;`

87 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 128ubyte, 0ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& medium_aquamarine() noexcept;`

88 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 102ubyte, 205ubyte, 170ubyte, 255ubyte }`.

`static const bgra_color& medium_blue() noexcept;`

89 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 0ubyte, 205ubyte, 255ubyte }`.

`static const bgra_color& medium_orchid() noexcept;`

90 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 186ubyte, 85ubyte, 211ubyte, 255ubyte }`.

`static const bgra_color& medium_purple() noexcept;`

91 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 147ubyte, 112ubyte, 219ubyte, 255ubyte }`.

`static const bgra_color& medium_sea_green() noexcept;`

92 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 60ubyte, 179ubyte, 113ubyte, 255ubyte }`.

`static const bgra_color& medium_slate_blue() noexcept;`

93 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 123ubyte, 104ubyte, 238ubyte, 255ubyte }`.

`static const bgra_color& medium_spring_green() noexcept;`

94 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 250ubyte, 154ubyte, 255ubyte }`.

`static const bgra_color& medium_turquoise() noexcept;`

95 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 72ubyte, 209ubyte, 204ubyte, 255ubyte }`.

`static const bgra_color& medium_violet_red() noexcept;`

96 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 199ubyte, 21ubyte, 133ubyte, 255ubyte }`.

`static const bgra_color& midnight_blue() noexcept;`

97 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 25ubyte, 25ubyte, 112ubyte, 255ubyte }`.

`static const bgra_color& mint_cream() noexcept;`

98 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 245ubyte, 255ubyte, 250ubyte, 255ubyte }`.

`static const bgra_color& misty_rose() noexcept;`

99 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 228ubyte, 225ubyte, 255ubyte }`.

`static const bgra_color& moccasin() noexcept;`

100 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 228ubyte, 181ubyte, 255ubyte }`.

`static const bgra_color& navajo_white() noexcept;`

101 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 222ubyte, 173ubyte, 255ubyte }`.

`static const bgra_color& navy() noexcept;`

102 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 0ubyte, 128ubyte, 255ubyte }`.

`static const bgra_color& old_lace() noexcept;`

103 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 253ubyte, 245ubyte, 230ubyte, 255ubyte }`.

`static const bgra_color& olive() noexcept;`

104 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 128ubyte, 128ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& olive_drab() noexcept;`

105 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 107ubyte, 142ubyte, 35ubyte, 255ubyte }`.

`static const bgra_color& orange() noexcept;`

106 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 165ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& orange_red() noexcept;`

107 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 69ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& orchid() noexcept;`

108 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 218ubyte, 112ubyte, 214ubyte, 255ubyte }`.

`static const bgra_color& pale_goldenrod() noexcept;`

109 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 238ubyte, 232ubyte, 170ubyte, 255ubyte }`.

`static const bgra_color& pale_green() noexcept;`

110 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 152ubyte, 251ubyte, 152ubyte, 255ubyte }`.

`static const bgra_color& pale_turquoise() noexcept;`

111 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 175ubyte, 238ubyte, 238ubyte, 255ubyte }`.

`static const bgra_color& pale_violet_red() noexcept;`

112 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 219ubyte, 112ubyte, 147ubyte, 255ubyte }`.

`static const bgra_color& papaya_whip() noexcept;`

113 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 239ubyte, 213ubyte, 255ubyte }`.

`static const bgra_color& peach_puff() noexcept;`

114 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 218ubyte, 185ubyte, 255ubyte }`.

`static const bgra_color& peru() noexcept;`

115 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 205ubyte, 133ubyte, 63ubyte, 255ubyte }`.

`static const bgra_color& pink() noexcept;`

116 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 192ubyte, 203ubyte, 255ubyte }`.

`static const bgra_color& plum() noexcept;`

117 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 221ubyte, 160ubyte, 221ubyte, 255ubyte }`.

`static const bgra_color& powder_blue() noexcept;`

118 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 176ubyte, 224ubyte, 230ubyte, 255ubyte }`.

`static const bgra_color& purple() noexcept;`

119 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 128ubyte, 0ubyte, 128ubyte, 255ubyte }`.

`static const bgra_color& red() noexcept;`

120 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 0ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& rosy_brown() noexcept;`

121 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 188ubyte, 143ubyte, 143ubyte, 255ubyte }`.

`static const bgra_color& royal_blue() noexcept;`

122 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 65ubyte, 105ubyte, 225ubyte, 255ubyte }`.

`static const bgra_color& saddle_brown() noexcept;`

123 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 139ubyte, 69ubyte, 19ubyte, 255ubyte }`.

`static const bgra_color& salmon() noexcept;`

124 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 250ubyte, 128ubyte, 114ubyte, 255ubyte }`.

`static const bgra_color& sandy_brown() noexcept;`

125 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 244ubyte, 164ubyte, 96ubyte, 255ubyte }`.

`static const bgra_color& sea_green() noexcept;`

126 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 46ubyte, 139ubyte, 87ubyte, 255ubyte }`.

`static const bgra_color& sea_shell() noexcept;`

127 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 245ubyte, 238ubyte, 255ubyte }`.

`static const bgra_color& sienna() noexcept;`

128 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 160ubyte, 82ubyte, 45ubyte, 255ubyte }`.

`static const bgra_color& silver() noexcept;`

129 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 192ubyte, 192ubyte, 192ubyte, 255ubyte }`.

`static const bgra_color& sky_blue() noexcept;`

130 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 135ubyte, 206ubyte, 235ubyte, 255ubyte }`.

```
static const bgra_color& slate_blue() noexcept;
```

131 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 106ubyte, 90ubyte, 205ubyte, 255ubyte }`.

```
static const bgra_color& slate_gray() noexcept;
```

132 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 112ubyte, 128ubyte, 144ubyte, 255ubyte }`.

```
static const bgra_color& slate_grey() noexcept;
```

133 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 112ubyte, 128ubyte, 144ubyte, 255ubyte }`.

```
static const bgra_color& snow() noexcept;
```

134 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 250ubyte, 250ubyte, 255ubyte }`.

```
static const bgra_color& spring_green() noexcept;
```

135 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 255ubyte, 127ubyte, 255ubyte }`.

```
static const bgra_color& steel_blue() noexcept;
```

136 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 70ubyte, 130ubyte, 180ubyte, 255ubyte }`.

```
static const bgra_color& tan() noexcept;
```

137 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 210ubyte, 180ubyte, 140ubyte, 255ubyte }`.

```
static const bgra_color& teal() noexcept;
```

138 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 128ubyte, 128ubyte, 255ubyte }`.

```
static const bgra_color& thistle() noexcept;
```

139 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 216ubyte, 191ubyte, 216ubyte, 255ubyte }`.

```
static const bgra_color& tomato() noexcept;
```

140 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 99ubyte, 71ubyte, 255ubyte }`.

```
static const bgra_color& transparent_black() noexcept;
```

141 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 0ubyte, 0ubyte, 0ubyte, 0ubyte }`.

```
static const bgra_color& turquoise() noexcept;
```

142 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 64ubyte, 244ubyte, 208ubyte, 255ubyte }`.

`static const bgra_color& violet() noexcept;`

143 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 238ubyte, 130ubyte, 238ubyte, 255ubyte }`.

`static const bgra_color& wheat() noexcept;`

144 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 245ubyte, 222ubyte, 179ubyte, 255ubyte }`.

`static const bgra_color& white() noexcept;`

145 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 255ubyte, 255ubyte, 255ubyte }`.

`static const bgra_color& white_smoke() noexcept;`

146 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 245ubyte, 245ubyte, 245ubyte, 255ubyte }`.

`static const bgra_color& yellow() noexcept;`

147 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 255ubyte, 255ubyte, 0ubyte, 255ubyte }`.

`static const bgra_color& yellow_green() noexcept;`

148 *Returns:* a const reference to the static `bgra_color` object `bgra_color{ 154ubyte, 205ubyte, 50ubyte, 255ubyte }`.

8.3.6 bgra_color non-member operators [bgracolor.ops]

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

1 *Returns:* `lhs.r() == rhs.r() && lhs.g() == rhs.g() && lhs.b() == rhs.b() && lhs.a() == rhs.a()`.

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

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

8.4 literals namespace [literals]

8.4.1 literals Synopsis [literals.synopsis]

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

8.4.2 literals operators [literals.operators]

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

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

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

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

9 Geometry [geometry]

9.1 Class vector_2d [vector2d]

9.1.1 vector_2d Description [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 double and a Y coordinate of type double.

9.1.2 vector_2d synopsis [vector2d.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class vector_2d {
  public:
    // 9.1.3, construct/copy/move/destroy:
    constexpr vector_2d() noexcept;
    constexpr vector_2d(double x, double y) noexcept;
    constexpr vector_2d(const vector_2d&) noexcept = default;
    constexpr vector_2d& operator=(const vector_2d&) noexcept = default;
    vector_2d(vector_2d&&) noexcept = default;
    vector_2d& operator=(vector_2d&&) noexcept = default;

    // 9.1.4, modifiers:
    constexpr void x(double value) noexcept;
    constexpr void y(double value) noexcept;

    // 9.1.5, observers:
    constexpr double x() const noexcept;
    constexpr double y() const noexcept;
    double magnitude() const noexcept;
    constexpr double magnitude_squared() const noexcept;
    constexpr double dot(const vector_2d& other) const noexcept;
    double angular_direction(const vector_2d& to) const noexcept
    vector_2d to_unit() const noexcept;

    // 9.1.6, member operators:
    vector_2d& operator+=(const vector_2d& rhs) noexcept;
    vector_2d& operator-=(const vector_2d& rhs) noexcept;
    vector_2d& operator*=(double rhs) noexcept;
  };

  // 9.1.7, 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, double rhs) noexcept;
  constexpr vector_2d operator*(double lhs, const vector_2d& rhs) noexcept;
} } } }
```

9.1.3 vector_2d constructors and assignment operators [vector2d.cons]

```
constexpr vector_2d() noexcept;
```

1 *Effects:* Constructs an object of type vector_2d.

2 The X coordinate shall be set to the value 0.0.

3 The Y coordinate shall be set to the value 0.0.

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

4 *Effects:* Constructs an object of type vector_2d.

5 The X coordinate shall be set to the value of x.

6 The Y coordinate shall be set to the value of y.

9.1.4 vector_2d modifiers [vector2d.modifiers]

```
constexpr void x(double value) noexcept;
```

1 *Effects:* The X coordinate shall be set to the value of x.

```
constexpr void y(double value) noexcept;
```

2 *Effects:* The Y coordinate shall be set to the value of y.

9.1.5 vector_2d observers [vector2d.observers]

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

1 *Returns:* The value of the X coordinate.

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

2 *Returns:* The value of the Y coordinate.

```
    double magnitude() const noexcept;
```

3 *Returns:* sqrt(*this.x() * *this.x() + *this.y() * *this.y()).

```
constexpr double magnitude_squared() const noexcept;
```

4 *Returns:* *this.x() * *this.x() + *this.y() * *this.y().

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

5 *Returns:* *this.x() * other.x() + *this.y() * other.y().

```
double angular_direction() const noexcept
```

6 *Returns:* The result of atan2(*this.y(), *this.x()) if it is greater than or equal to 0;0.

7 Otherwise, atan2(*this.y(), *this.x()) + two_pi<double>.

8 *Notes:* The purpose of adding 2π if the result is negative is to produce values in the range [0.0,2π).

```
vector_2d to_unit() const noexcept;
```

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

9.1.6 `vector_2d` member operators [vector2d.member.ops]

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

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

2 *Returns:* `*this`.

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

3 *Effects:* `*this = *this - rhs`.

4 *Returns:* `*this`.

```
vector_2d& operator*=(double rhs) noexcept;
```

5 *Effects:* `*this = *this * rhs`.

6 *Returns:* `*this`.

9.1.7 `vector_2d` non-member operators [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:* `vector_2d(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, double rhs) noexcept;
```

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

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

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

9.2 Class `rectangle` [rectangle]

9.2.1 `rectangle` Description [rectangle.intro]

1 The class `rectangle` describes a rectangle.

2 It has an X coordinate of type `double`, a Y coordinate of type `double`, a Width of type `double`, and a Height of type `double`.

9.2.2 rectangle synopsis [rectangle.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class rectangle {
  public:
    // 9.2.3, construct/copy/move/destroy:
    constexpr rectangle() noexcept;
    constexpr rectangle(double x, double y, double width, double height)
      noexcept;
    constexpr rectangle(const vector_2d& tl, const vector_2d& br) noexcept;

    // 9.2.4, modifiers:
    constexpr void x(double val) noexcept;
    constexpr void y(double val) noexcept;
    constexpr void width(double val) noexcept;
    constexpr void height(double val) noexcept;
    constexpr void top_left(const vector_2d& val) noexcept;
    constexpr void bottom_right(const vector_2d& val) noexcept;

    // 9.2.5, observers:
    constexpr double x() const noexcept;
    constexpr double y() const noexcept;
    constexpr double width() const noexcept;
    constexpr double height() const noexcept;
    constexpr vector_2d top_left() const noexcept;
    constexpr vector_2d bottom_right() const noexcept;
  };
} } } }
```

9.2.3 rectangle constructors [rectangle.cons]

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

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

2 The X coordinate, Y coordinate, Width, and Height shall each be set to the value `0.0`.

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

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

4 The X coordinate shall be set to the value of `x`.

5 The Y coordinate shall be set to the value of `y`.

6 The Width shall be set to the value of `w`.

7 The Height shall be set to the value of `h`.

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

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

9 The X coordinate shall be set to the value of `tl.x()`.

10 The Y coordinate shall be set to the value of `tl.y()`.

11 The Width shall be set to the value of `max(0.0, br.x() - tl.x())`.

12 The Height shall be set to the value of `max(0.0, br.y() - tl.y())`.

9.2.4 `rectangle` modifiers [rectangle.modifiers]

`constexpr void x(double val) noexcept;`

1 *Effects:* The X coordinate shall be set to the value of `val`.

`constexpr void y(double value) noexcept;`

2 *Effects:* The Y coordinate shall be set to the value of `val`.

`constexpr void width(double value) noexcept;`

3 *Effects:* The Width shall be set to the value of `val`.

`constexpr void height(double value) noexcept;`

4 *Effects:* The Height shall be set to the value of `val`.

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

5 *Effects:* The X coordinate shall be set to the value of `val.x()`.

Effects: The Y coordinate shall be set to the value of `val.y()`.

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

6 *Effects:* The Width shall be set to the value of `max(0.0, val.x() - *this.x())`.

7 The Height shall be set to the value of `max(0.0, value.y() - *this.y())`.

9.2.5 `rectangle` observers [rectangle.observers]

`constexpr double x() const noexcept;`

1 *Returns:* The value of the X coordinate.

`constexpr double y() const noexcept;`

2 *Returns:* The value of the Y coordinate.

`constexpr double width() const noexcept;`

3 *Returns:* The value of the Width.

`constexpr double height() const noexcept;`

4 *Returns:* The value of the Height.

`constexpr vector_2d top_left() const noexcept;`

5 *Returns:* A `vector_2d` object constructed from the value of the X coordinate as its first argument and the value of the Y coordinate as its second argument.

`constexpr vector_2d bottom_right() const noexcept;`

6 *Returns:* A `vector_2d` object constructed from the value of the Width added to the value of the X coordinate as its first argument and the value of the Height added to the value of the Y coordinate as its second argument.

9.3 Class `circle` [circle]

9.3.1 `circle` Description [circle.intro]

1 The class `circle` describes a circle.

2 It has a Center of type `vector_2d` and a Radius of type `double`.

9.3.2 circle synopsis [circle.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class circle {
  public:
    // 9.3.3, construct/copy/move/destroy:
    constexpr circle() noexcept;
    contexpr circle(const vector_2d& ctr, double rad) noexcept;

    // 9.3.4, modifiers:
    constexpr void center(const vector_2d& ctr) noexcept;
    constexpr void radius(double rad) noexcept;

    // 9.3.5, observers:
    constexpr vector_2d center() const noexcept;
    constexpr double radius() const noexcept;
  };
} } } }
```

9.3.3 circle constructors and assignment operators [circle.cons]

`constexpr circle() noexcept;`

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

2 The value of Center is `vector_2d{0,0, 0.0}`.

3 The value of Radius is `0.0`.

`constexpr circle(const vector_2d& ctr, double rad) noexcept;`

Requires: `rad >= 0.0`.

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

5 The value of Center is `ctr`.

6 The value of Radius is `rad`.

9.3.4 circle modifiers [circle.modifiers]

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

1 *Effects:* The value of Center is `ctr`.

`constexpr void radius(double rad) noexcept;`

Requires: `rad >= 0.0`.

2 *Effects:* The value of Radius is `rad`.

9.3.5 circle observers [circle.observers]

`constexpr double center() const noexcept;`

1 *Returns:* The value of Center.

`constexpr double radius() const noexcept;`

2 *Returns:* The value of Radius.

9.4 Class ellipse [ellipse]

9.4.1 ellipse Description [ellipse.intro]

1 The class `ellipse` describes a ellipse.

2 It has a Center of type `vector_2d`, an X Axis Radius of type `double`, and a Y Axis Radius of type `double`.

9.4.2 ellipse synopsis [ellipse.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class ellipse {
  public:
    // 9.4.3, construct/copy/move/destroy:
    constexpr ellipse() noexcept;
    constexpr ellipse(const vector_2d& ctr, double x, double y) noexcept;
    constexpr explicit ellipse(const circle& c) noexcept;

    // 9.4.4, modifiers:
    constexpr void center(const vector_2d& ctr) noexcept;
    constexpr void x_axis(double rad) noexcept;
    constexpr void y_axis(double rad) noexcept;

    // 9.4.5, observers:
    constexpr vector_2d center() const noexcept;
    constexpr double x_axis() const noexcept;
    constexpr double y_axis() const noexcept;
  };
} } } }
```

9.4.3 ellipse constructors and assignment operators [ellipse.cons]

`constexpr ellipse() noexcept;`

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

2 The value of Center is `vector_2d{0,0, 0.0}`.

3 The value of X Axis Radius is `0.0`.

4 The value of Y Axis Radius is `0.0`.

`constexpr ellipse(const vector_2d& ctr, double x, double y) noexcept;`

5 *Requires:* `x >= 0.0`.

6 `y >= 0.0`.

7 *Effects:* Constructs an object of type `ellipse`.

8 The value of Center is `ctr`.

9 The value of X Axis Radius is `x`.

10 The value of Y Axis Radius is `y`.

`constexpr explicit ellipse(const circle& c) noexcept;`

11 *Requires:* `c.radius() >= 0.0`.

12 *Effects:* Constructs an object of type `ellipse`.

13 The value of Center is `c.center()`.

14 The value of X Axis Radius is `c.radius()`.

15 The value of Y Axis Radius is `c.radius()`.

9.4.4 `ellipse` modifiers [ellipse.modifiers]

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

1 *Effects:* The value of Center is `ctr`.

```
constexpr void x_axis(double rad) noexcept;
```

Requires: `rad >= 0.0`.

2 *Effects:* The value of X Axis Radius is `rad`.

```
constexpr void y_axis(double rad) noexcept;
```

Requires: `rad >= 0.0`.

3 *Effects:* The value of Y Axis Radius is `rad`.

9.4.5 `ellipse` observers [ellipse.observers]

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

1 *Returns:* The value of Center.

```
constexpr double x_axis() const noexcept;
```

2 *Returns:* The value of X Axis Radius.

```
constexpr double y_axis() const noexcept;
```

3 *Returns:* The value of Y Axis Radius.

9.5 Class `matrix_2d` [matrix2d]

9.5.1 `matrix_2d` Description [matrix2d.intro]

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

2 Mathematically, regardless of the operations performed on a `matrix_2d`, the third column will always have the column vector value of $[0.0, 0.0, 1.0]$. As such, it is not included in the observable data of the matrix.

3 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.

4 If the third column's data members were observable, they would be:

(4.1) — M02, a `double` which would follow `m01` in the same row and would be assigned a value of `0.0`.

(4.2) — M12, a `double` which would follow `m11` in the same row and would be assigned a value of `0.0`.

(4.3) — M22, a `double` which would follow `m21` in the same row and would be assigned a value of `1.0`.

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

6 The values M00, M01, M10, M11, M20, M21 shall each be of type `double`.

9.5.2 matrix_2d synopsis [matrix2d.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class matrix_2d {
  public:
    // 9.5.3, construct/copy/move/destroy:
    constexpr matrix_2d() noexcept;
    constexpr matrix_2d(double v00, double v01, double v10, double v11,
      double v20, double v21) noexcept;

    // 9.5.4, static factory functions:
    constexpr static matrix_2d init_identity() noexcept;
    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(double radians) noexcept;
    constexpr static matrix_2d init_shear_x(double factor) noexcept;
    constexpr static matrix_2d init_shear_y(double factor) noexcept;

    // 9.5.5, modifiers:
    constexpr void m00(double v) noexcept;
    constexpr void m01(double v) noexcept;
    constexpr void m10(double v) noexcept;
    constexpr void m11(double v) noexcept;
    constexpr void m20(double v) noexcept;
    constexpr void m21(double v) noexcept;
    matrix_2d& translate(const vector_2d& v) noexcept;
    matrix_2d& scale(const vector_2d& v) noexcept;
    matrix_2d& rotate(double radians) noexcept;
    matrix_2d& shear_x(double factor) noexcept;
    matrix_2d& shear_y(double factor) noexcept;
    matrix_2d& invert();

    // 9.5.6, observers:
    constexpr double m00() const noexcept;
    constexpr double m01() const noexcept;
    constexpr double m10() const noexcept;
    constexpr double m11() const noexcept;
    constexpr double m20() const noexcept;
    constexpr double m21() const noexcept;
    constexpr bool is_finite() const noexcept;
    constexpr bool is_invertible() const noexcept;
    constexpr double determinant() const;
    constexpr vector_2d transform_point(const vector_2d& pt) const noexcept;

    // 9.5.7, matrix_2d member operators:
    matrix_2d& operator*=(const matrix_2d& rhs) noexcept;
  };

  // 9.5.8, 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;
} } } }
```

9.5.3 `matrix_2d` constructors and assignment operators [matrix2d.cons]

```
constexpr matrix_2d() noexcept;
```

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

2 The M00 value shall be set to `1.0`.

3 The M01 value shall be set to `0.0`.

4 The M10 value shall be set to `0.0`.

5 The M11 value shall be set to `1.0`.

6 The M20 value shall be set to `0.0`.

7 The M21 value shall be set to `0.0`.

8 *Note:* The resulting matrix is the identity matrix, which can also be obtained from `matrix_2d::init_identity()`.

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

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

10 The M00 value shall be set to the value of `v00`.

11 The M01 value shall be set to the value of `v01`.

12 The M10 value shall be set to the value of `v10`.

13 The M11 value shall be set to the value of `v11`.

14 The M20 value shall be set to the value of `v20`.

15 The M21 value shall be set to the value of `v21`.

9.5.4 `matrix_2d` static factory functions [matrix2d.staticfactories]

```
constexpr static matrix_2d init_identity() noexcept;
```

1 *Returns:* `matrix(1.0, 0.0, 0.0, 1.0, 0.0, 0.0)`.

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

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

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

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

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

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

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

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

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

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

9.5.5 `matrix_2d` modifiers [matrix2d.modifiers]

`constexpr void m00(double val) noexcept;`

1 *Effects:* The M00 value shall be set to the value of `val`.

`constexpr void m01(double val) noexcept;`

2 *Effects:* The M01 value shall be set to the value of `val`.

`constexpr void m10(double val) noexcept;`

3 *Effects:* The M10 value shall be set to the value of `val`.

`constexpr void m11(double val) noexcept;`

4 *Effects:* The M11 value shall be set to the value of `val`.

`constexpr void m20(double val) noexcept;`

5 *Effects:* The M20 value shall be set to the value of `val`.

`constexpr void m21(double value) noexcept;`

6 *Effects:* The M21 value shall be set to the value of `val`.

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

7 *Effects:* `*this = init_translate(value) * (*this)`.

8 *Returns:* `*this`.

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

9 *Effects:* `*this = init_scale(value) * (*this)`.

10 *Returns:* `*this`.

`matrix_2d& rotate(double radians) noexcept;`

11 *Effects:* `*this = init_rotate(radians) * (*this)`.

12 *Returns:* `*this`.

`matrix_2d& shear_x(double factor) noexcept;`

13 *Effects:* `*this = init_shear_x(factor) * (*this)`.

14 *Returns:* `*this`.

`matrix_2d& shear_y(double factor) noexcept;`

15 *Effects:* `*this = init_shear_y(factor) * (*this)`.

16 *Returns:* `*this`.

`matrix_2d& invert() noexcept;`

17 *Requires:* `*this.isfinite() == true`.

18 *Effects:*

```
    const auto det = *this.m00() * *this.m11() - *this.m01() * *this.m10*();
    const auto inverseDet = 1.0 / det;

    const auto cM02 = 0.0;
    const auto cM12 = 0.0;
    const auto cM22 = 1.0;

    const auto adjugateM00 =   *this.m11() * cM22 - cM12 * *this.m21();
    const auto adjugateM01 = -(*this.m01() * cM22 - cM02 * *this.m21());
    const auto adjugateM10 = -(*this.m10() * cM22 - cM12 * *this.m20());
    const auto adjugateM11 =   *this.m00() * cM22 - cM02 * *this.m20();
    const auto adjugateM20 =   *this.m10() * *this.m21() - *this.m11() *
      *this.m20();
    const auto adjugateM21 = -(*this.m00() * *this.m21() - *this.m01() *
      *this.m20());

    *this.m00(inverseDet * adjugateM00);
    *this.m01(inverseDet * adjugateM01);
    *this.m10(inverseDet * adjugateM10);
    *this.m11(inverseDet * adjugateM11);
    *this.m20(inverseDet * adjugateM20);
    *this.m21(inverseDet * adjugateM21);
```

19 *Returns:* `*this`.

9.5.6 `matrix_2d` observers [matrix2d.observers]

`constexpr double m00() const noexcept;`

1 *Returns:* The M00 value.

`constexpr double m01() const noexcept;`

2 *Returns:* The M01 value.

`constexpr double m10() const noexcept;`

3 *Returns:* The M10 value.

`constexpr double m11() const noexcept;`

4 *Returns:* The M11 value.

`constexpr double m20() const noexcept;`

5 *Returns:* The M20 value.

`constexpr double m21() const noexcept;`

6 *Returns:* The M21 value.

`constexpr bool is_finite const noexcept;`

7 *Returns:* `true` if all of the following are true:

(7.1) — `isfinite(*this.m00())`

(7.2) — `isfinite(*this.m01())`

(7.3) — `isfinite(*this.m10())`

(7.4) — `isfinite(*this.m11())`

(7.5) — `isfinite(*this.m20())`

(7.6) — `isfinite(*this.m21())`

8 Otherwise returns `false`.

```
constexpr bool is_invertible() const noexcept;
```

9 *Requires:* `*this.is_finite() == true`.

10 *Returns:* `*this.m00() * *this.m11() - *this.m01() * *this.m10()) != 0.0`.

```
constexpr double determinant() const noexcept;
```

11 *Requires:* `*this.is_finite() == true`.

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

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

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

9.5.7 matrix_2d member operators [matrix2d.member.ops]

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

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

2 *Returns:* `*this`

9.5.8 matrix_2d non-member operators [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)`.

10 Paths [paths]

1 Paths define geometric objects which can be stroked (Table 17), filled, masked, and used to define a Clip Area (12.5.1.

2 Paths are created using a `path_builder` object, which stores a path group.

3 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 (12.18.1) 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. For example, a pixel count expressed as 1280x720 means that there are 1280 horizontal pixels per row of pixels and 720 vertical pixels per column of pixels for a total of 921600 pixels.

4 A path may contain degenerate path segments because of special rules, e.g. the `line_cap` state of a `surface` object when they are rendered, which are set forth below.

5 A `path_group` object is an immutable resource wrapper containing a path group (10.20). A `path_group` object is created from a `path_builder` object. It can also be default constructed, in which case the `path_group` object contains no paths.

10.1 Processing paths [paths.processing]

1 This section is normative.

2 It describes how to convert a properly formed path group into a *processed path group*. The steps required to create a processed path group require the existence of certain state data:

Table 2 — path group processing state data

Data	Type	Initial Value
Transformation Matrix	`matrix_2d`	`matrix_2d::init_identity()`.
Origin	`vector_2d`	`vector_2d{ }`.
Current Point	`optional<vector_2d>`	`optional<vector_2d>{ }`.

3 Certain path instructions and path segments will modify this state data. The state data is used to ensure that the coordinates of all points in the path group's paths are properly transformed to their intended coordinates based on the effects of those path instructions and path segments.

4 [*Note:* The coordinates of a processed path group are in whatever units the user desires. This coordinate space is known as the User Coordinate Space (12.16.4). Path instructions such as `path_data::change_matrix` and `path_data::change_origin` affect the interpretation of path items that follow them within the path group. When a rendering and composing operation takes place, the coordinates of points within the processed path group are transformed into coordinates in the Surface Coordinate Space (12.16.4) using the Surface Properties' (12.16.3.4) Surface Matrix (12.1.1). — *end note*]

5 The source code below demonstrates how to properly convert a path group into a processed path group.

6 The `process_path_data` function template transforms the path group contained in a `path_builder` class template object into a processed path group which is returned as a `vector<path_data::path_data_types>` object. The processed path group only contains `path_data::abs_move`, `path_data::abs_line`, `path_data::abs_cubic_curve`, and `path_data::close_path` path items.

7 [*Note:* If the underlying rendering and presentation technologies do not support lines or cubic Bézier curves,

Bresenham's algorithms and variations and improvements upon them allow computation of appropriate pixel values for these primitives. — *end note*]

```
#include <cmath>
#include <vector>
#include <variant>
#include <experimental/io2d>"

namespace process_path {
  using namespace ::std;
  using namespace experimental::io2d;

  enum class abs_cubic_curve_sfinae {};
  constexpr abs_cubic_curve_sfinae abs_cubic_curve_sfinae_val = {};
  enum class abs_ellipse_sfinae {};
  constexpr static abs_ellipse_sfinae abs_ellipse_sfinae_val = {};
  enum class abs_line_sfinae {};
  constexpr abs_line_sfinae abs_line_sfinae_val = {};
  enum class abs_move_sfinae {};
  constexpr abs_move_sfinae abs_move_sfinae_val = {};
  enum class abs_quadratic_curve_sfinae {};
  constexpr abs_quadratic_curve_sfinae abs_quadratic_curve_sfinae_val = {};
  enum class abs_rectangle_sfinae {};
  constexpr static abs_rectangle_sfinae abs_rectangle_sfinae_val = {};
  enum class arc_clockwise_sfinae {};
  constexpr arc_clockwise_sfinae arc_clockwise_sfinae_val = {};
  enum class arc_counterclockwise_sfinae {};
  constexpr arc_counterclockwise_sfinae arc_counterclockwise_sfinae_val = {};
  enum class change_matrix_sfinae {};
  constexpr change_matrix_sfinae change_matrix_sfinae_val = {};
  enum class change_origin_sfinae {};
  constexpr change_origin_sfinae change_origin_sfinae_val = {};
  enum class close_path_sfinae {};
  constexpr close_path_sfinae close_path_sfinae_val = {};
  enum class new_path_sfinae {};
  constexpr new_path_sfinae new_path_sfinae_val = {};
  enum class rel_cubic_curve_sfinae {};
  constexpr rel_cubic_curve_sfinae rel_cubic_curve_sfinae_val = {};
  enum class rel_ellipse_sfinae {};
  constexpr static rel_ellipse_sfinae rel_ellipse_sfinae_val = {};
  enum class rel_line_sfinae {};
  constexpr rel_line_sfinae rel_line_sfinae_val = {};
  enum class rel_move_sfinae {};
  constexpr rel_move_sfinae rel_move_sfinae_val = {};
  enum class rel_quadratic_curve_sfinae {};
  constexpr rel_quadratic_curve_sfinae rel_quadratic_curve_sfinae_val = {};
  enum class rel_rectangle_sfinae {};
  constexpr static rel_rectangle_sfinae rel_rectangle_sfinae_val = {};

  template <class Allocator>
  vector<path_data::path_data_types> process_path_data(
    const path_builder<Allocator>& pf);

  template <class _TItem>
  struct path_factory_process_visit {
```

```
constexpr static double twoThirds = 2.0 / 3.0;

template <class T, enable_if_t<is_same_v<T, path_data::abs_cubic_curve>,
  abs_cubic_curve_sfinae> = abs_cubic_curve_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  auto pt1 = m.transform_point(item.control_point_1() - origin) + origin;
  auto pt2 = m.transform_point(item.control_point_2() - origin) + origin;
  auto pt3 = m.transform_point(item.end_point() - origin) + origin;
  if (!currentPoint.has_value()) {
    currentPoint = item.control_point_1();
    v.emplace_back(in_place_type<path_data::abs_move>, pt1);
    closePoint = pt1;
  }
  v.emplace_back(in_place_type<path_data::abs_cubic_curve>, pt1,
    pt2, pt3);
  currentPoint = item.end_point();
}
template <class T, enable_if_t<is_same_v<T,
  path_data::abs_ellipse>, abs_ellipse_sfinae> = abs_ellipse_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>&v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  const auto m2 = m;
  const auto o2 = origin;
  currentPoint.reset();
  path_factory_process_visit<path_data::change_origin>::template
    perform(path_data::change_origin{ item.center() }, v, m, origin,
    currentPoint, closePoint);
  path_factory_process_visit<path_data::change_matrix>::template
    perform(path_data::change_matrix{ matrix_2d::init_scale({
    item.x_axis() / item.y_axis(), 1.0 }) * m }, v, m, origin,
    currentPoint, closePoint);
  path_factory_process_visit<path_data::arc_clockwise>::template
    perform(path_data::arc_clockwise{ item.center(), item.y_axis(), 0.0,
    two_pi<double> }, v, m, origin, currentPoint, closePoint);
  path_factory_process_visit<path_data::change_matrix>::template
    perform(path_data::change_matrix{ m2 }, v, m, origin, currentPoint,
    closePoint);
  path_factory_process_visit<path_data::change_origin>::template
    perform(path_data::change_origin{ o2 }, v, m, origin, currentPoint,
    closePoint);
}
template <class T, enable_if_t<is_same_v<T, path_data::abs_line>,
  abs_line_sfinae> = abs_line_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  if (currentPoint.has_value()) {
    currentPoint = item.to();
    auto pt = m.transform_point(currentPoint.value() - origin) + origin;
    v.emplace_back(in_place_type<path_data::abs_line>, pt);
  }
  else {
    currentPoint = item.to();
    auto pt = m.transform_point(currentPoint.value() - origin) + origin;
    v.emplace_back(in_place_type<path_data::abs_move>, pt);
    v.emplace_back(in_place_type<path_data::abs_line>, pt);
    closePoint = pt;
  }
}
template <class T, enable_if_t<is_same_v<T, path_data::abs_move>,
  abs_move_sfinae> = abs_move_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  currentPoint = item.to();
  auto pt = m.transform_point(currentPoint.value() - origin) + origin;
  v.emplace_back(in_place_type<path_data::abs_move>, pt);
  closePoint = pt;
}
template <class T, enable_if_t<is_same_v<T,
  path_data::abs_quadratic_curve>, abs_quadratic_curve_sfinae> =
  abs_quadratic_curve_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  // Turn it into a cubic curve since cairo doesn’t have quadratic curves.
  vector_2d beginPt;
  auto controlPt = m.transform_point(item.control_point() - origin) +
    origin;
  auto endPt = m.transform_point(item.end_point() - origin) + origin;
  if (!currentPoint.has_value()) {
    currentPoint = item.control_point();
    v.emplace_back(in_place_type<path_data::abs_move>, controlPt);
    closePoint = controlPt;
    beginPt = controlPt;
  }
  else {
    beginPt = m.transform_point(currentPoint.value() - origin) + origin;
  }
  vector_2d cpt1 = { ((controlPt.x() - beginPt.x()) * twoThirds) +
    beginPt.x(), ((controlPt.y() - beginPt.y()) * twoThirds) +
    beginPt.y() };
  vector_2d cpt2 = { ((controlPt.x() - endPt.x()) * twoThirds) +
    endPt.x(), ((controlPt.y() - endPt.y()) * twoThirds) + endPt.y() };
  v.emplace_back(in_place_type<path_data::abs_cubic_curve>, cpt1, cpt2,
    endPt);
  currentPoint = item.end_point();
}
template <class T, enable_if_t<is_same_v<T,
  path_data::abs_rectangle>, abs_rectangle_sfinae> =
  abs_rectangle_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>&v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  path_factory_process_visit::template perform(path_data::abs_move{ {
    item.x(), item.y() } }, v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::rel_line{ {
    item.width(), 0.0 } }, v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::rel_line{ {
    0.0, item.height() } }, v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::rel_line{ {
    -item.width(), 0.0 } }, v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::close_path{ {
    item.x(), item.y() } }, v, m, origin, currentPoint, closePoint);
}
template <class T, enable_if_t<is_same_v<T, path_data::arc_clockwise>,
  arc_clockwise_sfinae> = arc_clockwise_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  {
    auto ctr = item.center();
    auto rad = item.radius();
    auto ang1 = item.angle_1();
    auto ang2 = item.angle_2();
    while (ang2 < ang1) {
      ang2 += two_pi<double>;
    }
    vector_2d pt0, pt1, pt2, pt3;
    int bezCount = 1;
    double theta = ang2 - ang1;
    double phi{};
    while (theta >= half_pi<double>) {
      theta /= 2.0;
      bezCount += bezCount;
    }
    phi = theta / 2.0;
    auto cosPhi = cos(phi);
    auto sinPhi = sin(phi);
    pt0.x(cosPhi);
    pt0.y(-sinPhi);
    pt3.x(pt0.x());
    pt3.y(-pt0.y());
    pt1.x((4.0 - cosPhi) / 3.0);
    pt1.y(-(((1.0 - cosPhi) * (3.0 - cosPhi)) / (3.0 * sinPhi)));
    pt2.x(pt1.x());
    pt2.y(-pt1.y());
    phi = -phi;
    auto rotCwFn = [](const vector_2d& pt, double a) -> vector_2d {
      return { pt.x() * cos(a) + pt.y() * sin(a),
        -(pt.x() * -(sin(a)) + pt.y() * cos(a)) };
    };
    pt0 = rotCwFn(pt0, phi);
    pt1 = rotCwFn(pt1, phi);
    pt2 = rotCwFn(pt2, phi);
    pt3 = rotCwFn(pt3, phi);

    auto currTheta = ang1;
    const auto startPt =
      ctr + rotCwFn({ pt0.x() * rad, pt0.y() * rad }, currTheta);
    if (currentPoint.has_value()) {
      currentPoint = startPt;
      auto pt = m.transform_point(currentPoint.value() - origin) + origin;
      v.emplace_back(in_place_type<path_data::abs_line>, pt);
    }
    else {
      currentPoint = startPt;
      auto pt = m.transform_point(currentPoint.value() - origin) + origin;
      v.emplace_back(in_place_type<path_data::abs_move>, pt);
      closePoint = pt;
    }
    for (; bezCount > 0; bezCount--) {
      auto cpt1 = ctr + rotCwFn({ pt1.x() * rad, pt1.y() * rad },
      currTheta);
      auto cpt2 = ctr + rotCwFn({ pt2.x() * rad, pt2.y() * rad },
        currTheta);
      auto cpt3 = ctr + rotCwFn({ pt3.x() * rad, pt3.y() * rad },
        currTheta);
      currentPoint = cpt3;
      cpt1 = m.transform_point(cpt1 - origin) + origin;
      cpt2 = m.transform_point(cpt2 - origin) + origin;
      cpt3 = m.transform_point(cpt3 - origin) + origin;
      v.emplace_back(in_place_type<path_data::abs_cubic_curve>, cpt1,
        cpt2, cpt3);
      currTheta += theta;
    }
  }
}
template <class T, enable_if_t<is_same_v<T,
  path_data::arc_counterclockwise>, arc_counterclockwise_sfinae> =
  arc_counterclockwise_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  {
    auto ctr = item.center();
    auto rad = item.radius();
    auto ang1 = item.angle_1();
    auto ang2 = item.angle_2();
    while (ang2 > ang1) {
      ang2 -= two_pi<double>;
    }
    vector_2d pt0, pt1, pt2, pt3;
    int bezCount = 1;
    double theta = ang1 - ang2;
    double phi{};
    while (theta >= half_pi<double>) {
      theta /= 2.0;
      bezCount += bezCount;
    }
    phi = theta / 2.0;
    auto cosPhi = cos(phi);
    auto sinPhi = sin(phi);
    pt0.x(cosPhi);
    pt0.y(-sinPhi);
    pt3.x(pt0.x());
    pt3.y(-pt0.y());
    pt1.x((4.0 - cosPhi) / 3.0);
    pt1.y(-(((1.0 - cosPhi) * (3.0 - cosPhi)) / (3.0 * sinPhi)));
    pt2.x(pt1.x());
    pt2.y(-pt1.y());
    auto rotCwFn = [](const vector_2d& pt, double a) -> vector_2d {
      return { pt.x() * cos(a) + pt.y() * sin(a),
        -(pt.x() * -(sin(a)) + pt.y() * cos(a)) };
    };
    pt0 = rotCwFn(pt0, phi);
    pt1 = rotCwFn(pt1, phi);
    pt2 = rotCwFn(pt2, phi);
    pt3 = rotCwFn(pt3, phi);
    auto shflPt = pt3;
    pt3 = pt0;
    pt0 = shflPt;
    shflPt = pt2;
    pt2 = pt1;
    pt1 = shflPt;
    auto currTheta = ang1;
    const auto startPt =
      ctr + rotCwFn({ pt0.x() * rad, pt0.y() * rad }, currTheta);
    if (currentPoint.has_value()) {
      currentPoint = startPt;
      auto pt = m.transform_point(currentPoint.value() - origin) + origin;
      v.emplace_back(in_place_type<path_data::abs_line>, pt);
    }
    else {
      currentPoint = startPt;
      auto pt = m.transform_point(currentPoint.value() - origin) + origin;
      v.emplace_back(in_place_type<path_data::abs_move>, pt);
      closePoint = pt;
    }
    for (; bezCount > 0; bezCount--) {
      auto cpt1 = ctr + rotCwFn({ pt1.x() * rad, pt1.y() * rad },
        currTheta);
      auto cpt2 = ctr + rotCwFn({ pt2.x() * rad, pt2.y() * rad },
        currTheta);
      auto cpt3 = ctr + rotCwFn({ pt3.x() * rad, pt3.y() * rad },
        currTheta);
      currentPoint = cpt3;
      cpt1 = m.transform_point(cpt1 - origin) + origin;
      cpt2 = m.transform_point(cpt2 - origin) + origin;
      cpt3 = m.transform_point(cpt3 - origin) + origin;
      v.emplace_back(in_place_type<path_data::abs_cubic_curve>, cpt1,
        cpt2, cpt3);
      currTheta -= theta;
    }
  }
}
template <class T, enable_if_t<is_same_v<T, path_data::change_matrix>,
  change_matrix_sfinae> = change_matrix_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>&,
  matrix_2d& m, vector_2d&, optional<vector_2d>&, vector_2d&) {
  if (!m.is_finite()) {
    throw system_error(make_error_code(io2d_error::invalid_matrix));
  }
  if (!m.is_invertible()) {
    throw system_error(make_error_code(io2d_error::invalid_matrix));
    return;
  }
  m = item.matrix();
}
template <class T, enable_if_t<is_same_v<T, path_data::change_origin>,
  change_origin_sfinae> = change_origin_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>&,
  matrix_2d&, vector_2d& origin, optional<vector_2d>&, vector_2d&) {
  origin = item.origin();
}
template <class T, enable_if_t<is_same_v<T,
  path_data::close_path>, close_path_sfinae> = close_path_sfinae_val>
static void perform(const T&, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  if (currentPoint.has_value()) {
    v.emplace_back(in_place_type<path_data::close_path>, closePoint);
    v.emplace_back(in_place_type<path_data::abs_move>,
      closePoint);
    if (!m.is_finite() || !m.is_invertible()) {
      throw system_error(make_error_code(io2d_error::invalid_matrix));
    }
    auto invM = matrix_2d{ m }.invert();
    // Need to assign the untransformed closePoint value to currentPoint.
    currentPoint = invM.transform_point(closePoint - origin) + origin;
  }
}
template <class T, enable_if_t<is_same_v<T, path_data::new_path>,
  new_path_sfinae> = new_path_sfinae_val>
static void perform(const T&, vector<path_data::path_data_types>&,
  matrix_2d&, vector_2d&, optional<vector_2d>& currentPoint, vector_2d&) {
  currentPoint.reset();
}
template <class T, enable_if_t<is_same_v<T, path_data::rel_cubic_curve>,
  rel_cubic_curve_sfinae> = rel_cubic_curve_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d&) {
  if (!currentPoint.has_value()) {
    throw system_error(make_error_code(io2d_error::invalid_path_data));
  }
  auto pt1 = m.transform_point(item.control_point_1() +
    currentPoint.value() - origin) + origin;
  auto pt2 = m.transform_point(item.control_point_2() +
    currentPoint.value() - origin) + origin;
  auto pt3 = m.transform_point(item.end_point() + currentPoint.value() -
    origin) + origin;
  v.emplace_back(in_place_type<path_data::abs_cubic_curve>,
    pt1, pt2, pt3);
  currentPoint = item.end_point() + currentPoint.value();
}
template <class T, enable_if_t<is_same_v<T, path_data::rel_ellipse>,
  rel_ellipse_sfinae> = rel_ellipse_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>&v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  if (!currentPoint.has_value()) {
    throw system_error(make_error_code(io2d_error::invalid_path_data));
  }
  const auto m2 = m;
  const auto o2 = origin;
  const auto cpt2 = currentPoint;
  currentPoint.reset();
  path_factory_process_visit::template perform(path_data::change_origin{
    item.center() + cpt2.value() }, v, m, origin, currentPoint,
    closePoint);
  path_factory_process_visit::template perform(path_data::change_matrix{
    matrix_2d::init_scale({ item.x_axis() / item.y_axis(), 1.0 }) * m },
    v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::arc_clockwise{
    item.center() + cpt2.value(), item.y_axis(), 0.0, two_pi<double> },
    v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::change_matrix{
    m2 }, v, m, origin, currentPoint, closePoint);
  path_factory_process_visit::template perform(path_data::change_origin{
    o2 }, v, m, origin, currentPoint, closePoint);
}
template <class T, enable_if_t<is_same_v<T, path_data::rel_line>,
  rel_line_sfinae> = rel_line_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d&) {
  if (!currentPoint.has_value()) {
    throw system_error(make_error_code(io2d_error::invalid_path_data));
  }
  currentPoint = item.to() + currentPoint.value();
  auto pt = m.transform_point(currentPoint.value() - origin) + origin;
  v.emplace_back(in_place_type<path_data::abs_line>, pt);
}
template <class T, enable_if_t<is_same_v<T, path_data::rel_move>,
  rel_move_sfinae> = rel_move_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
  vector_2d& closePoint) {
  if (!currentPoint.has_value()) {
    throw system_error(make_error_code(io2d_error::invalid_path_data));
  }
  currentPoint = item.to() + currentPoint.value();
  auto pt = m.transform_point(currentPoint.value() - origin) + origin;
  v.emplace_back(in_place_type<path_data::abs_move>, pt);
  closePoint = pt;
}
template <class T, enable_if_t<is_same_v<T,
  path_data::rel_quadratic_curve>, rel_quadratic_curve_sfinae> =
  rel_quadratic_curve_sfinae_val>
static void perform(const T& item, vector<path_data::path_data_types>& v,
  matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
      vector_2d&) {
      if (!currentPoint.has_value()) {
        throw system_error(make_error_code(io2d_error::invalid_path_data));
      }
      // Turn it into a cubic curve since cairo doesn’t have quadratic curves.
      vector_2d beginPt;
      auto controlPt = m.transform_point(item.control_point() +
        currentPoint.value() - origin) + origin;
      auto endPt = m.transform_point(item.end_point() + currentPoint.value() -
        origin) + origin;
      beginPt = m.transform_point(currentPoint.value() - origin) + origin;
      vector_2d cpt1 = { ((controlPt.x() - beginPt.x()) * twoThirds) +
        beginPt.x(), ((controlPt.y() - beginPt.y()) * twoThirds) +
        beginPt.y() };
      vector_2d cpt2 = { ((controlPt.x() - endPt.x()) * twoThirds) +
        endPt.x(), ((controlPt.y() - endPt.y()) * twoThirds) + endPt.y() };
      v.emplace_back(in_place_type<path_data::abs_cubic_curve>, cpt1, cpt2,
        endPt);
      currentPoint = item.end_point() + currentPoint.value();
    }
    template <class T, enable_if_t<is_same_v<T, path_data::rel_rectangle>,
      rel_rectangle_sfinae> = rel_rectangle_sfinae_val>
    static void perform(const T& item, vector<path_data::path_data_types>&v,
      matrix_2d& m, vector_2d& origin, optional<vector_2d>& currentPoint,
      vector_2d& closePoint) {
      path_factory_process_visit::template perform(path_data::rel_move{ {
        item.x(), item.y() } }, v, m, origin, currentPoint, closePoint);
      path_factory_process_visit::template perform(path_data::rel_line{ {
        item.width(), 0.0 } }, v, m, origin, currentPoint, closePoint);
      path_factory_process_visit::template perform(path_data::rel_line{ {
        0.0, item.height() } }, v, m, origin, currentPoint, closePoint);
      path_factory_process_visit::template perform(path_data::rel_line{ {
        -item.width(), 0.0 } }, v, m, origin, currentPoint, closePoint);
      path_factory_process_visit::template perform(path_data::close_path{ {
        item.x(), item.y() } }, v, m, origin, currentPoint, closePoint);
    }
  };

  template <class Allocator>
  inline vector<path_data::path_data_types> process_path_data(
    const path_builder<Allocator>& pf) {
    matrix_2d m;
    vector_2d origin;
    // Tracks the untransformed current point.
    optional<vector_2d> currentPoint = optional<vector_2d>{};
    vector_2d closePoint;   // Tracks the transformed close point.
    vector<path_data::path_data_types> v;

    for (const path_data::path_data_types& val : pf) {
      visit([&m, &origin, &currentPoint, &closePoint, &v](auto&& item) {
        using T = remove_cv_t<remove_reference_t<decltype(item)>>;
        path_factory_process_visit<T>::template perform<T>(item, v, m,
          origin, currentPoint, closePoint);
      }, val);
    }
```

```
      return v;
    }
  }
```

10.2 Class abs_cubic_curve [abscubiccurve]

1 The class abs_cubic_curve describes a path segment that is a cubic Bézier curve.

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.

10.2.1 abs_cubic_curve synopsis [abscubiccurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_cubic_curve {
    public:
      // 10.2.2, construct:
      constexpr abs_cubic_curve() noexcept;
      constexpr abs_cubic_curve(const vector_2d& cpt1, const vector_2d& cpt2,
        const vector_2d& ept) noexcept;

      // 10.2.3, modifiers:
      constexpr void control_point_1(const vector_2d& cpt) noexcept;
      constexpr void control_point_2(const vector_2d& cpt) noexcept;
      constexpr void end_point(const vector_2d& ept) noexcept;


      // 10.2.4, observers:
      constexpr vector_2d control_point_1() const noexcept;
      constexpr vector_2d control_point_2() const noexcept;
      constexpr vector_2d end_point() const noexcept;
    };
  };
} } } }
```

10.2.2 abs_cubic_curve constructors [abscubiccurve.cons]

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

1 *Effects:* Constructs an object of type abs_cubic_curve.

2 The first control point shall be set to the value of vector_2d{0.0, 0.0}.

3 The second control point shall be set to the value of vector_2d{0.0, 0.0}.

4 The end point shall be set to the value of vector_2d{0.0, 0.0}.

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

5 *Effects:* Constructs an object of type abs_cubic_curve.

6 The first control point shall be set to the value of cpt1.

7 The second control point shall be set to the value of cpt2.

8 The end point shall be set to the value of ept.

10.2.3 abs_cubic_curve modifiers [abscubiccurve.modifiers]

`constexpr void control_point_1(const vector_2d& cpt) noexcept;`

1 *Effects:* The first control point shall be set to the value of `cpt`.

`constexpr void control_point_2(const vector_2d& cpt) noexcept;`

2 *Effects:* The second control point shall be set to the value of `cpt`.

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

3 *Effects:* The end point shall be set to the value of `ept`.

10.2.4 abs_cubic_curve observers [abscubiccurve.observers]

`constexpr vector_2d control_point_1() const noexcept;`

1 *Returns:* The value of the first control point.

`constexpr vector_2d control_point_2() const noexcept;`

2 *Returns:* The value of the second control point.

`constexpr vector_2d end_point() const noexcept;`

3 *Returns:* The value of the end point.

10.3 Class abs_ellipse [absellipse]

1 The class `abs_ellipse` describes a path instruction that add a new path consisting of an ellipse and closes it.

2 It has a Center of type `vector_2d`, an X Axis Radius of type `double`, and a Y Axis Radius of type `double`.

10.3.1 abs_ellipse synopsis [absellipse.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_ellipse {
    public:
      // 10.3.2, construct/copy/move/destroy:
      constexpr abs_ellipse() noexcept;
      constexpr abs_ellipse(const vector_2d& ctr, double x, double y) noexcept;
      constexpr explicit abs_ellipse(const circle& c) noexcept;

      // 10.3.3, modifiers:
      constexpr void center(const vector_2d& ctr) noexcept;
      constexpr void x_axis(double rad) noexcept;
      constexpr void y_axis(double rad) noexcept;

      // 10.3.4, observers:
      constexpr vector_2d center() const noexcept;
      constexpr double x_axis() const noexcept;
      constexpr double y_axis() const noexcept;
    };
  }
} } } }
```

10.3.2 abs_ellipse constructors [absellipse.cons]

```
constexpr abs_ellipse() noexcept;
```

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

2 The value of Center is `vector_2d{0,0, 0.0}`.

3 The value of X Axis Radius is `0.0`.

4 The value of Y Axis Radius is `0.0`.

```
constexpr abs_ellipse(const vector_2d& ctr, double x, double y) noexcept;
```

5 *Requires:* `x >= 0.0`.

6 `y >= 0.0`.

7 *Effects:* Constructs an object of type `abs_ellipse`.

8 The value of Center is `ctr`.

9 The value of X Axis Radius is `x`.

10 The value of Y Axis Radius is `y`.

```
constexpr explicit abs_ellipse(const circle& c) noexcept;
```

11 *Requires:* `c.radius() >= 0.0`.

12 *Effects:* Constructs an object of type `abs_ellipse`.

13 The value of Center is `c.center()`.

14 The value of X Axis Radius is `c.radius()`.

15 The value of Y Axis Radius is `c.radius()`.

10.3.3 `abs_ellipse` modifiers [absellipse.modifiers]

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

1 *Effects:* The value of Center is `ctr`.

```
constexpr void x_axis(double rad) noexcept;
```

Requires: `rad >= 0.0`.

2 *Effects:* The value of X Axis Radius is `rad`.

```
constexpr void y_axis(double rad) noexcept;
```

Requires: `rad >= 0.0`.

3 *Effects:* The value of Y Axis Radius is `rad`.

10.3.4 `abs_ellipse` observers [absellipse.observers]

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

1 *Returns:* The value of Center.

```
constexpr double x_axis() const noexcept;
```

2 *Returns:* The value of X Axis Radius.

```
constexpr double y_axis() const noexcept;
```

3 *Returns:* The value of Y Axis Radius.

10.4 Class abs_line [absline]

1 The class abs_line describes a path segment that is a line.

2 It has an end point of type vector_2d.

10.4.1 abs_line synopsis [absline.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_line {
    public:
      // 10.4.2, construct:
      constexpr abs_line() noexcept;
      constexpr explicit abs_line(const vector_2d& pt) noexcept;

      // 10.4.3, modifiers:
      constexpr void to(const vector_2d& pt) noexcept;

      // 10.4.4, observers:
      constexpr vector_2d to() const noexcept;
    };
  };
} } } }
```

10.4.2 abs_line constructors and assignment operators [absline.cons]

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

1 *Effects:* Constructs an object of type abs_line.

2 The end point shall be set to the value of vector_2d{0.0, 0.0}.

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

3 *Effects:* Constructs an object of type abs_line.

4 The end point shall be set to the value of pt.

10.4.3 abs_line modifiers [absline.modifiers]

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

1 *Effects:* The end point shall be set to the value of pt.

10.4.4 abs_line observers [absline.observers]

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

1 *Returns:* The value of the end point.

10.5 Class abs_move [absmove]

1 The class abs_move describes a path operation that creates a new path and makes the previous path, if any, an open path unless it was closed by close_path.

2 It has an end point of type vector_2d.

3 The end point is also the start point of the new path and its last-move-to point.

10.5.1 abs_move synopsis [absmove.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_move {
    public:
      // 10.5.2, construct:
      constexpr abs_move() noexcept;
      constexpr explicit abs_move(const vector_2d& pt) noexcept;

      // 10.5.3, modifiers:
      constexpr void to(const vector_2d& pt) noexcept;

      // 10.5.4, observers:
      constexpr vector_2d to() const noexcept;
    };
  };
} } } }
```

10.5.2 abs_move constructors [absmove.cons]

`constexpr abs_move() noexcept;`

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

2 The end point shall be set to the value `vector_2d{0.0, 0.0}`.

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

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

4 The end point shall be set to the value of `pt`.

10.5.3 abs_move modifiers [absmove.modifiers]

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

1 *Effects:* The end point shall be set to the value of `pt`.

10.5.4 abs_move observers [absmove.observers]

`constexpr vector_2d to() const noexcept;`

1 *Returns:* The value of the end point.

10.6 Class abs_quadratic_curve [absquadraticcurve]

1 The class `abs_quadratic_curve` describes a path segment that is a quadratic Bézier curve.

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

10.6.1 abs_quadratic_curve synopsis [absquadraticcurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_cubic_curve {
    public:
      // 10.6.2, construct:
      constexpr abs_quadratic_curve() noexcept;
      constexpr abs_quadratic_curve(const vector_2d& cpt, const vector_2d& ept)
        noexcept;
      // 10.6.3, modifiers:
      constexpr void control_point(const vector_2d& cpt) noexcept;
      constexpr void end_point(const vector_2d& ept) noexcept;

      // 10.6.4, observers:
      constexpr vector_2d control_point() const noexcept;
      constexpr vector_2d end_point() const noexcept;
    };
  };
} } } }
```

10.6.2 abs_quadratic_curve constructors [absquadraticcurve.cons]

`constexpr abs_quadratic_curve() noexcept;`

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

2 The control point shall be set to the value of `vector_2d{0.0, 0.0}`.

3 The end point shall be set to the value of `vector_2d{0.0, 0.0}`.

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

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

5 The control point shall be set to the value of `cpt`.

6 The end point shall be set to the value of `ept`.

10.6.3 abs_quadratic_curve modifiers [absquadraticcurve.modifiers]

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

1 *Effects:* The control point shall be set to the value of `cpt`.

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

2 *Effects:* The end point shall be set to the value of `ept`.

10.6.4 abs_quadratic_curve observers [absquadraticcurve.observers]

`constexpr vector_2d control_point() const noexcept;`

1 *Returns:* The value of the control point.

`constexpr vector_2d end_point() const noexcept;`

2 *Returns:* The value of the end point.

10.7 Class abs_rectangle [absrectangle]

1 The class `abs_rectangle` describes a path instruction that adds a rectangle to the current path.

2 It has an X coordinate of type `double`, a Y coordinate of type `double`, a Width of type `double`, and a Height of type `double`.

10.7.1 abs_rectangle synopsis [absrectangle.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_rectangle {
    public:
      // 10.7.2, constructors:
      constexpr abs_rectangle() noexcept;
      constexpr abs_rectangle(double x, double y, double w, double h) noexcept;
      constexpr abs_rectangle(const vector_2d& tl, const vector_2d& br)
        noexcept;
      constexpr abs_rectangle(const rectangle& r);

      // 10.7.3, modifiers:
      constexpr void x(double value) noexcept;
      constexpr void y(double value) noexcept;
      constexpr void width(double value) noexcept;
      constexpr void height(double value) noexcept;
      constexpr void top_left(const vector_2d& value) noexcept;
      constexpr void bottom_right(const vector_2d& value) noexcept;
      constexpr void top_left_bottom_right(const vector_2d& tl,
        const vector_2d& br) noexcept;

      // 10.7.4, observers:
      constexpr double x() const noexcept;
      constexpr double y() const noexcept;
      constexpr double width() const noexcept;
      constexpr double height() const noexcept;
      constexpr double left() const noexcept;
      constexpr double right() const noexcept;
      constexpr double top() const noexcept;
      constexpr double bottom() const noexcept;
      constexpr vector_2d top_left() const noexcept;
      constexpr vector_2d bottom_right() const noexcept;
    };
  }
} } } }
```

10.7.2 abs_rectangle constructors [absrectangle.cons]

`constexpr abs_rectangle() noexcept;`

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

2 The X coordinate, Y coordinate, Width, and Height shall each be set to the value `0.0`.

`constexpr abs_rectangle(double x, double y, double w, double h) noexcept;`

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

4 The X coordinate shall be set to the value of `x`.

5 The Y coordinate shall be set to the value of `y`.

6 The Width shall be set to the value of `w`.

7 The Height shall be set to the value of `h`.

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

8 *Effects:* Constructs an object of type `abs_rectangle`.

9 The X coordinate shall be set to the value of `tl.x()`.

10 The Y coordinate shall be set to the value of `tl.y()`.

11 The Width shall be set to the value of `max(0.0, br.x() - tl.x())`.

12 The Height shall be set to the value of `max(0.0, br.y() - tl.y())`.

10.7.3 `abs_rectangle` modifiers [absrectangle.modifiers]

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

1 *Effects:* The X coordinate shall be set to the value of `val`.

```
constexpr void y(double value) noexcept;
```

2 *Effects:* The Y coordinate shall be set to the value of `val`.

```
constexpr void width(double value) noexcept;
```

3 *Effects:* The Width shall be set to the value of `val`.

```
constexpr void height(double value) noexcept;
```

4 *Effects:* The Height shall be set to the value of `val`.

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

5 *Effects:* The X coordinate shall be set to the value of `val.x()`.

Effects: The Y coordinate shall be set to the value of `val.y()`.

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

6 *Effects:* The Width shall be set to the value of `max(0.0, val.x() - *this.x())`.

7 The Height shall be set to the value of `max(0.0, value.y() - *this.y())`.

10.7.4 `abs_rectangle` observers [absrectangle.observers]

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

1 *Returns:* The value of the X coordinate.

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

2 *Returns:* The value of the Y coordinate.

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

3 *Returns:* The value of the Width.

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

4 *Returns:* The value of the Height.

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

5 *Returns:* A `vector_2d` object constructed from the value of the X coordinate as its first argument and the value of the Y coordinate as its second argument.

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

6 *Returns:* A `vector_2d` object constructed from the value of the Width added to the value of the X coordinate as its first argument and the value of the Height added to the value of the Y coordinate as its second argument.

10.8 Class `arc_clockwise` [arcclockwise]

1 The class `arc_clockwise` describes a path segment that is a circular arc with clockwise rotation.

2 It has a Circle of type `circle`, a First Angle of type `double`, and a Second Angle of type `double`.

3 The values for the First Angle and Second Angle are in radians.

4 [*Note:* Although the value of the First Angle may be greater than the value of the Second Angle, when processed as described in 10.1, `two_pi<double>` is added to the Second Angle until the value of the Second Angle is greater than or equal to the value of the First Angle. — *end note*]

10.8.1 `arc_clockwise` synopsis [arcclockwise.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class arc_clockwise {
    public:
      // 10.8.2, construct/copy/move/destroy:
      constexpr arc_clockwise() noexcept;
      constexpr arc_clockwise(const experimental::io2d::circle& c,
        double angle1, double angle2) noexcept;
      constexpr arc_clockwise(const vector_2d& ctr, double rad,
        double angle1, double angle2) noexcept;

      // 10.8.3, modifiers:
      constexpr void circle(const experimental::io2d::circle& c) noexcept;
      constexpr void center(const vector_2d& ctr) noexcept;
      constexpr void radius(double r) noexcept;
      constexpr void angle_1(double radians) noexcept;
      constexpr void angle_2(double radians) noexcept;

      // 10.8.4, observers:
      constexpr experimental::io2d::circle circle() const noexcept;
      constexpr vector_2d center() const noexcept;
      constexpr double radius() const noexcept;
      constexpr double angle_1() const noexcept;
      constexpr double angle_2() const noexcept;
    };
  };
} } } }
```

10.8.2 `arc_clockwise` constructors and assignment operators [arcclockwise.cons]

```
constexpr arc_clockwise() noexcept;
```

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

2 The Circle shall be set to the value of `experimental::io2d::circle{ }`.

3 The First Angle shall be set to the value of `0.0`.

4 The Second Angle shall be set to the value of `0.0`.

```
constexpr arc_clockwise(const experimental::io2d::circle& c, double angle1,
  double angle2) noexcept;
```

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

6 The Circle shall be set to the value of `c`.

7 The First Angle shall be set to the value of `angle1`.

8 The Second Angle shall be set to the value of `angle2`.

```
constexpr arc_clockwise(const vector_2d& ctr, double rad, double angle1,
  double angle2) noexcept;
```

9 *Effects:* Constructs an object of type `arc_clockwise`.

10 The Circle's Center (9.3.1) shall be set to the value of `ctr`.

11 The Circle's Radius (9.3.1) shall be set to the value of `rad`.

12 The First Angle shall be set to the value of `angle1`.

13 The Second Angle shall be set to the value of `angle2`.

10.8.3 `arc_clockwise` modifiers [arcclockwise.modifiers]

```
constexpr void circle(const experimental::io2d::circle& c) noexcept;
```

1 *Effects:* The Circle shall be set to the value of `c`.

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

2 *Effects:* The Circle's Center (9.3.1) shall be set to the value of `ctr`.

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

3 *Effects:* The Circle's Radius (9.3.1) shall be set to the value of `r`.

```
constexpr void angle_1(double radians) noexcept;
```

4 *Effects:* The First Angle shall be set to the value of `radians`.

```
constexpr void angle_2(double radians) noexcept;
```

5 *Effects:* The Second Angle shall be set to the value of `radians`.

10.8.4 `arc_clockwise` observers [arcclockwise.observers]

```
constexpr experimental::io2d::circle circle() const noexcept;
```

1 *Returns:* The value of the Circle.

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

2 *Returns:* The value of the Circle's Center (9.3.1).

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

3 *Returns:* The value of the Circle's Radius (9.3.1).

```
constexpr double angle_1() const noexcept;
```

4 *Returns:* The value of the First Angle.

```
constexpr double angle_2() const noexcept;
```

5 *Returns:* The value of the Second Angle.

10.9 Class `arc_counterclockwise` [arccounterclockwise]

1 The class `arc_counterclockwise` describes a path segment that is a circular arc with counterclockwise rotation.

2 It has a Circle of type `circle`, a First Angle of type `double`, and a Second Angle of type `double`.

3 The values for the First Angle and Second Angle are in radians.

4 [*Note:* Although the value of the Second Angle may be greater than the value of the First Angle, when processed as described in 10.1, `two_pi<double>` is subtracted from the Second Angle until the value of the First Angle is greater than or equal to the value of the Second Angle. — *end note*]

10.9.1 `arc_counterclockwise` synopsis [arccounterclockwise.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class arc_counterclockwise {
    public:
      // 10.9.2, construct:
      constexpr arc_counterclockwise() noexcept;
      constexpr arc_counterclockwise(const experimental::io2d::circle& c,
        double angle1, double angle2) noexcept;
      constexpr arc_counterclockwise(const vector_2d& ctr, double rad,
        double angle1, double angle2) noexcept;

      // 10.9.3, modifiers:
      constexpr void circle(const experimental::io2d::circle& c) noexcept;
      constexpr void center(const vector_2d& ctr) noexcept;
      constexpr void radius(double r) noexcept;
      constexpr void angle_1(double radians) noexcept;
      constexpr void angle_2(double radians) noexcept;

      // 10.9.4, observers:
      constexpr experimental::io2d::circle circle() const noexcept;
      constexpr vector_2d center() const noexcept;
      constexpr double radius() const noexcept;
      constexpr double angle_1() const noexcept;
      constexpr double angle_2() const noexcept;
    };
  };
} } } }
```

10.9.2 `arc_counterclockwise` constructors and assignment operators [arccounterclockwise.cons]

```
constexpr arc_counterclockwise() noexcept;
```

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

2 The Circle shall be set to the value of `experimental::io2d::circle{ }`.

3 The First Angle shall be set to the value of `0.0`.

4 The Second Angle shall be set to the value of `0.0`.

```
constexpr arc_counterclockwise(const experimental::io2d::circle& c,
  double angle1, double angle2) noexcept;
```

5 *Effects:* Constructs an object of type `arc_counterclockwise`. arc_countercl6ockwise The Circle shall be set to the value of `c`.

7 The First Angle shall be set to the value of `angle1`.

8 The Second Angle shall be set to the value of `angle2`.

```
constexpr arc_counterclockwise(const vector_2d& ctr, double rad, double angle1,
  double angle2) noexcept;
```

9 *Effects:* Constructs an object of type `arc_counterclockwise`.

10 The Circle's Center (9.3.1) shall be set to the value of `ctr`.

11 The Circle's Radius (9.3.1) shall be set to the value of `rad`.

12 The First Angle shall be set to the value of `angle1`.

13 The Second Angle shall be set to the value of `angle2`.

10.9.3 arc_counterclockwise modifiers [arccounterclockwise.modifiers]

```
constexpr void circle(const experimental::io2d::circle& c) noexcept;
```

1 *Effects:* The Circle shall be set to the value of `c`.

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

2 *Effects:* The Circle's Center (9.3.1) shall be set to the value of `ctr`.

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

3 *Effects:* The Circle's Radius (9.3.1) shall be set to the value of `r`.

```
constexpr void angle_1(double radians) noexcept;
```

4 *Effects:* The First Angle shall be set to the value of `radians`.

```
constexpr void angle_2(double radians) noexcept;
```

5 *Effects:* The Second Angle shall be set to the value of `radians`.

10.9.4 arc_counterclockwise observers [arccounterclockwise.observers]

```
constexpr experimental::io2d::circle circl() const noexcept;
```

1 *Returns:* The value of the Circle.

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

2 *Returns:* The value of the Circle's Center (9.3.1).

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

3 *Returns:* The value of the Circle's Radius (9.3.1).

```
constexpr double angle_1() const noexcept;
```

4 *Returns:* The value of the First Angle.

```
constexpr double angle_2() const noexcept;
```

5 *Returns:* The value of the Second Angle.

10.10 Class change_matrix [changematrix]

10.10.1 change_matrix synopsis [changematrix.synopsis]

1 The class change_matrix describes path instruction that changes the transformation matrix used in processing a path group as described by 10.1.

2 It has a Matrix of type matrix_2d.

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class change_matrix {
    public:
      // 10.10.2, construct:
      constexpr change_matrix() noexcept;
      constexpr explicit change_matrix(const matrix_2d& m) noexcept;

      // 10.10.3, modifiers:
      constexpr void matrix(const matrix_2d& m) noexcept;

      // 10.10.4, observers:
      constexpr matrix_2d matrix() const noexcept;
    };
  };
} } } }
```

10.10.2 change_matrix constructors and assignment operators [changematrix.cons]

`constexpr change_matrix() noexcept;`

1 *Effects:* Constructs an object of type change_matrix.

2 The Matrix shall be set to the value of matrix{ }.

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

3 *Effects:* Constructs an object of type change_matrix.

4 The Matrix shall be set to the value of m.

10.10.3 change_matrix modifiers [changematrix.modifiers]

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

1 *Effects:* The Matrix shall be set to the value of m.

10.10.4 change_matrix observers [changematrix.observers]

`constexpr matrix_2d matrix() const noexcept;`

1 *Returns:* The value of the Matrix.

10.11 Class change_origin [changeorigin]

1 The class change_origin describes path instruction that changes the origin used in processing a path group as described by 10.1.

2 It has an Origin of type vector_2d.

10.11.1 change_origin synopsis [changeorigin.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class change_origin {
    public:
      // 10.11.2, construct:
      constexpr change_origin() noexcept;
      constexpr explicit change_origin(const vector_2d& pt) noexcept;

      // 10.11.3, modifiers:
      constexpr void origin(const vector_2d& pt) noexcept;

      // 10.11.4, observers:
      constexpr vector_2d origin() const noexcept;
    };
  };
} } } }
```

10.11.2 change_origin constructors and assignment operators [changeorigin.cons]

`constexpr change_origin() noexcept;`

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

2 The Origin shall be set to the value of `vector_2d{ }`.

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

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

4 The Origin shall be set to the value of `pt`.

10.11.3 change_origin modifiers [changeorigin.modifiers]

`constexpr void origin(const vector_2d& value) noexcept;`

1 *Effects:* The Origin shall be set to the value of `pt`.

10.11.4 change_origin observers [changeorigin.observers]

`constexpr vector_2d origin() const noexcept;`

1 *Returns:* The value of the Origin.

10.12 Class close_path [closepath]

1 This class is a path instruction that creates a closed path within a path group.

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

10.12.1 close_path synopsis [closepath.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class close_path {
      // 10.12.2, construct:
      constexpr close_path() noexcept;
      constexpr explicit close_path(const vector_2d& to) noexcept;

      // 10.12.3, modifiers:
      constexpr void to(const vector_2d& value) noexcept;

      // 10.12.4, observers:
      constexpr vector_2d to() const noexcept;
    };
  };
} } } }
```

10.12.2 close_path constructors [closepath.cons]

`constexpr close_path() noexcept;`

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

2 The end point shall be set to the value of `vector_2d{}`.

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

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

4 The end point shall be set to the value of `pt`.

10.12.3 abs_move modifiers [closepath.modifiers]

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

1 *Effects:* The end point shall be set to the value of `pt`.

10.12.4 abs_move observers [closepath.observers]

`constexpr vector_2d to() const noexcept;`

1 *Returns:* The value of the end point.

10.13 Class new_path [newpath]

1 The class `new_path` describes a path operation that creates a new path and makes the previous path, if any, an open path unless it was closed by `close_path`.

2 The new path has no current point.

10.13.1 new_path synopsis [newpath.synopsis]

```
  namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    namespace path_data {
      class new_path {
        constexpr new_path() noexcept;
      };
    };
  } } } }
```

10.14 Class rel_cubic_curve [relcubiccurve]

1 The class `rel_cubic_curve` describes a path segment that is a cubic Bézier curve.

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`.

3 All of its points are relative to the most recently established current point.

10.14.1 rel_cubic_curve synopsis [relcubiccurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_cubic_curve {
    public:
      // 10.14.2, construct
      constexpr rel_cubic_curve() noexcept;
      constexpr rel_cubic_curve(const vector_2d& cpt1, const vector_2d& cpt2,
        const vector_2d& ept) noexcept;

      // 10.14.3, modifiers:
      constexpr void control_point_1(const vector_2d& cpt) noexcept;
      constexpr void control_point_2(const vector_2d& cpt) noexcept;
      constexpr void end_point(const vector_2d& ept) noexcept;

      // 10.14.4, observers:
      constexpr vector_2d control_point_1() const noexcept;
      constexpr vector_2d control_point_2() const noexcept;
      constexpr vector_2d end_point() const noexcept;
    };
  };
} } } }
```

10.14.2 rel_cubic_curve constructors [relcubiccurve.cons]

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

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

2 The first control point shall be set to the value of `vector_2d{0.0, 0.0}`.

3 The second control point shall be set to the value of `vector_2d{0.0, 0.0}`.

4 The end point shall be set to the value of `vector_2d{0.0, 0.0}`.

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

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

6 The first control point shall be set to the value of `cpt1`.

7 The second control point shall be set to the value of `cpt2`.

8 The end point shall be set to the value of `ept`.

10.14.3 rel_cubic_curve modifiers [relcubiccurve.modifiers]

```
constexpr void control_point_1(const vector_2d& cpt) noexcept;
```

1 *Effects:* The first control point shall be set to the value of `cpt`.

```
constexpr void control_point_2(const vector_2d& cpt) noexcept;
```

2 *Effects:* The second control point shall be set to the value of `cpt`.

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

3 *Effects:* The end point shall be set to the value of `ept`.

10.14.4 rel_cubic_curve observers [relcubiccurve.observers]

```
constexpr vector_2d control_point_1() const noexcept;
```

1 *Returns:* The value of the first control point.

```
constexpr vector_2d control_point_2() const noexcept;
```

2 *Returns:* The value of the second control point.

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

3 *Returns:* The value of the end point.

10.15 Class rel_ellipse [relellipse]

1 The class `rel_ellipse` describes a path instruction that adds an ellipse to the current path.

2 It has a Center of type `vector_2d`, an X Axis Radius of type `double`, and a Y Axis Radius of type `double`.

10.15.1 abs_ellipse synopsis [relellipse.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_ellipse {
    public:
      // 10.15.2, construct/copy/move/destroy:
      constexpr rel_ellipse() noexcept;
      constexpr rel_ellipse(const vector_2d& ctr, double x, double y) noexcept;
      constexpr explicit rel_ellipse(const circle& c) noexcept;

      // 10.15.3, modifiers:
      constexpr void center(const vector_2d& ctr) noexcept;
      constexpr void x_axis(double rad) noexcept;
      constexpr void y_axis(double rad) noexcept;

      // 10.15.4, observers:
      constexpr vector_2d center() const noexcept;
      constexpr double x_axis() const noexcept;
      constexpr double y_axis() const noexcept;
    };
  }
} } } }
```

10.15.2 rel_ellipse constructors [relellipse.cons]

```
constexpr rel_ellipse() noexcept;
```

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

2 The value of Center is `vector_2d{0,0, 0.0}`.

3 The value of X Axis Radius is `0.0`.

4 The value of Y Axis Radius is `0.0`.

```
constexpr rel_ellipse(const vector_2d& ctr, double x, double y) noexcept;
```

5 *Requires:* `x >= 0.0`.

6 `y >= 0.0`.

7 *Effects:* Constructs an object of type `rel_ellipse`.

8 The value of Center is `ctr`.

9 The value of X Axis Radius is `x`.

10 The value of Y Axis Radius is `y`.

```
constexpr explicit rel_ellipse(const circle& c) noexcept;
```

11 *Requires:* `c.radius() >= 0.0`.

12 *Effects:* Constructs an object of type `rel_ellipse`.

13 The value of Center is `c.center()`.

14 The value of X Axis Radius is `c.radius()`.

15 The value of Y Axis Radius is `c.radius()`.

10.15.3 `rel_ellipse` modifiers [relellipse.modifiers]

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

1 *Effects:* The value of Center is `ctr`.

```
constexpr void x_axis(double rad) noexcept;
```

Requires: `rad >= 0.0`.

2 *Effects:* The value of X Axis Radius is `rad`.

```
constexpr void y_axis(double rad) noexcept;
```

Requires: `rad >= 0.0`.

3 *Effects:* The value of Y Axis Radius is `rad`.

10.15.4 `rel_ellipse` observers [relellipse.observers]

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

1 *Returns:* The value of Center.

```
constexpr double x_axis() const noexcept;
```

2 *Returns:* The value of X Axis Radius.

```
constexpr double y_axis() const noexcept;
```

3 *Returns:* The value of Y Axis Radius.

10.16 Class `rel_line` [relline]

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

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

3 Its end point is relative to the most recently established current point.

10.16.1 rel_line synopsis [relline.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_line {
    public:
      // 10.16.2, construct:
      constexpr rel_line() noexcept;
      constexpr explicit rel_line(const vector_2d& pt) noexcept;

      // 10.16.3, modifiers:
      constexpr void to(const vector_2d& pt) noexcept;

      // 10.16.4, observers:
      constexpr vector_2d to() const noexcept;
    };
  };
} } } }
```

10.16.2 rel_line constructors [relline.cons]

`constexpr rel_line() noexcept;`

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

2 The end point shall be set to the value of `vector_2d{0.0, 0.0}`.

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

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

4 The end point shall be set to the value of `pt`.

10.16.3 rel_line modifiers [relline.modifiers]

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

1 *Effects:* The end point shall be set to the value of `pt`.

10.16.4 rel_line observers [relline.observers]

`constexpr vector_2d to() const noexcept;`

1 *Returns:* The value of the end point.

10.17 Class rel_move [relmove]

1 The class `rel_move` describes a path operation that creates a new path and makes the previous path, if any, an open path unless it was closed by `close_path`.

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

3 Its end point is relative to the most recently established current point.

4 The relative end point is also the start point of the new path and its last-move-to point.

10.17.1 rel_move synopsis [relmove.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_move {
    public:
      // 10.17.2, construct:
      constexpr rel_move() noexcept;
      constexpr explicit rel_move(const vector_2d& pt) noexcept;

      // 10.17.3, modifiers:
      constexpr void to(const vector_2d& pt) noexcept;

      // 10.17.4, observers:
      constexpr vector_2d to() const noexcept;
    };
  };
} } } }
```

10.17.2 `rel_move` constructors [relmove.cons]

`constexpr rel_move() noexcept;`

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

2 The end point shall be set to the value `vector_2d{0.0, 0.0}`.

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

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

4 The end point shall be set to the value of `pt`.

10.17.3 `rel_move` modifiers [relmove.modifiers]

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

1 *Effects:* The end point shall be set to the value of `pt`.

10.17.4 `rel_move` observers [relmove.observers]

`constexpr vector_2d to() const noexcept;`

1 *Returns:* The value of the end point.

10.18 Class `rel_quadratic_curve` [relquadraticcurve]

1 The class `rel_quadratic_curve` describes a path segment that is a quadratic Bézier curve.

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

3 All of its points are relative to the most recently established current point.

10.18.1 `rel_quadratic_curve` synopsis [relquadraticcurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_cubic_curve {
    public:
      // 10.18.2, construct:
      constexpr rel_quadratic_curve() noexcept;
      constexpr rel_quadratic_curve(const vector_2d& cpt, const vector_2d& ept)
        noexcept;

      // 10.18.3, modifiers:
      constexpr void control_point(const vector_2d& cpt) noexcept;
      constexpr void end_point(const vector_2d& ept) noexcept;
      // 10.18.4, observers:
      constexpr vector_2d control_point() const noexcept;
      constexpr vector_2d end_point() const noexcept;
    };
  };
} } } }
```

10.18.2 rel_quadratic_curve constructors [relquadraticcurve.cons]

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

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

2 The control point shall be set to the value of `cpt`.

3 The end point shall be set to the value of `ept`.

10.18.3 rel_quadratic_curve modifiers [relquadraticcurve.modifiers]

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

1 *Effects:* The control point shall be set to the value of `cp`.

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

2 *Effects:* The end point shall be set to the value of `ept`.

10.18.4 rel_quadratic_curve observers [relquadraticcurve.observers]

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

1 *Returns:* The value of the control point.

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

2 *Returns:* The value of the end point.

10.19 Class rel_rectangle [relrectangle]

1 The class `rel_rectangle` describes a path instruction that adds a rectangle to the current path.

2 It has an X coordinate of type `double`, a Y coordinate of type `double`, a Width of type `double`, and a Height of type `double`.

10.19.1 rel_rectangle synopsis [relrectangle.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_rectangle {
    public:
      // 10.19.2, constructors:
      constexpr rel_rectangle() noexcept;
      constexpr rel_rectangle(double x, double y, double w, double h) noexcept;
      constexpr rel_rectangle(const vector_2d& tl, const vector_2d& br)
        noexcept;
      constexpr rel_rectangle(const rectangle& r);

      // 10.19.3, modifiers:
      constexpr void x(double value) noexcept;
```

```
        constexpr void y(double value) noexcept;
        constexpr void width(double value) noexcept;
        constexpr void height(double value) noexcept;
        constexpr void top_left(const vector_2d& value) noexcept;
        constexpr void bottom_right(const vector_2d& value) noexcept;
        constexpr void top_left_bottom_right(const vector_2d& tl,
          const vector_2d& br) noexcept;

        // 10.19.4, observers:
        constexpr double x() const noexcept;
        constexpr double y() const noexcept;
        constexpr double width() const noexcept;
        constexpr double height() const noexcept;
        constexpr double left() const noexcept;
        constexpr double right() const noexcept;
        constexpr double top() const noexcept;
        constexpr double bottom() const noexcept;
        constexpr vector_2d top_left() const noexcept;
        constexpr vector_2d bottom_right() const noexcept;
      };
    }
  } } } }
```

10.19.2 `rel_rectangle` constructors [relrectangle.cons]

`constexpr rel_rectangle() noexcept;`

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

2 The X coordinate, Y coordinate, Width, and Height shall each be set to the value `0.0`.

`constexpr rel_rectangle(double x, double y, double w, double h) noexcept;`

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

4 The X coordinate shall be set to the value of `x`.

5 The Y coordinate shall be set to the value of `y`.

6 The Width shall be set to the value of `w`.

7 The Height shall be set to the value of `h`.

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

8 *Effects:* Constructs an object of type `rel_rectangle`.

9 The X coordinate shall be set to the value of `tl.x()`.

10 The Y coordinate shall be set to the value of `tl.y()`.

11 The Width shall be set to the value of `max(0.0, br.x() - tl.x())`.

12 The Height shall be set to the value of `max(0.0, br.y() - tl.y())`.

10.19.3 `rel_rectangle` modifiers [relrectangle.modifiers]

`constexpr void x(double val) noexcept;`

1 *Effects:* The X coordinate shall be set to the value of `val`.

`constexpr void y(double value) noexcept;`

2 *Effects:* The Y coordinate shall be set to the value of `val`.

```
constexpr void width(double value) noexcept;
```

3 *Effects:* The Width shall be set to the value of `val`.

```
constexpr void height(double value) noexcept;
```

4 *Effects:* The Height shall be set to the value of `val`.

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

5 *Effects:* The X coordinate shall be set to the value of `val.x()`.

Effects: The Y coordinate shall be set to the value of `val.y()`.

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

6 *Effects:* The Width shall be set to the value of `max(0.0, val.x() - *this.x())`.

7 The Height shall be set to the value of `max(0.0, value.y() - *this.y())`.

10.19.4 `rel_rectangle` observers [relrectangle.observers]

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

1 *Returns:* The value of the X coordinate.

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

2 *Returns:* The value of the Y coordinate.

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

3 *Returns:* The value of the Width.

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

4 *Returns:* The value of the Height.

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

5 *Returns:* A `vector_2d` object constructed from the value of the X coordinate as its first argument and the value of the Y coordinate as its second argument.

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

6 *Returns:* A `vector_2d` object constructed from the value of the Width added to the value of the X coordinate as its first argument and the value of the Height added to the value of the Y coordinate as its second argument.

10.20 Class `path_group` [pathgroup]

1 The class `path_group` contains the result of processing 10.1 a `path_builder` object. How it stores the resulting data is unspecified.

2 A `path_group` object is used by a `surface`-derived object for rendering and composing operations.

3 The contents of a `path_group` object are immutable, however the contents it contains are changeable using copy assignment or move assignment.

4 A `path_group` object can be default constructed. Default construction of a `path_group` object produces the same result as constructing it from an empty `path_builder` object.

10.20.1 path_group synopsis [pathgroup.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class path_group {
    public:
    // 10.20.2, construct/copy/destroy:
    explicit path_group(const path_builder& pb);
    path_group(const path_builder& pb, error_code& ec) noexcept;
  };
} } } }
```

10.20.2 path_group constructors [pathgroup.cons]

```
explicit path_group(const path_builder& pb);
path_group(const path_builder& pb, error_code& ec) noexcept;
```

1 *Effects:* Constructs an object of class `path_group`. When a `path_group` object is used by a `surface`-derived object, it shall produce the same result as-if its contents were constructed by processing the `path_builder` object as specified at 10.1.

2 *Throws:* As specified in Error reporting (5).

3 *Remarks:* A `path_group` object may require further processing by the graphics subsystem when it is passed as an argument to a `surface` or `surface`-derived object.

4 Implementations should avoid or minimize the need for further processing of a `path_group` object after it has been constructed.

5 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

6 `io2d_error::no_current_point` if, when processing the path group of the `path_builder`, an operation was encountered which required a current point the current point had no value.

7 `io2d_error::invalid_matrix` if, when processing path group of the `path_builder`, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

8 `io2d_error::invalid_status` if the implementation or graphics subsystem encountered an error other than those specified above.

10.21 Class path_builder [pathbuilder]

1 The class `path_builder` is a container that stores and manipulates objects of type `path_data::data` 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::data` objects in a path builder is referred to as its path group.

4 In addition to its path group, a path builder has an origin of type `vector_2d`, a transformation matrix of type `matrix_2d`, a current point of type `std::optional<vector_2d>`, and a last-move-to point of type `vector_2d`.

5 When a path builder is default constructed:

(5.1) — The path group shall be empty.

(5.2) — The current point shall not contain a value.

(5.3) — The transformation matrix shall have a value of `matrix_2d::init_identity{ }`.

(5.4) — The origin shall have a value of `vector_2d{ }`.

10.21.1 path_builder synopsis [pathbuilder.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  template <class Allocator = allocator<path_data::path_data_types>>
  class path_builder {
  public:
    using value_type = path_data::path_data_types;
    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>;

    // 10.21.3, 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;

    // 10.21.6, 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;

// 10.21.4, 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;

// 10.21.5, modifiers:
void new_path() noexcept;
void close_path() noexcept;
void arc_clockwise(const vector_2d& center, double radius, double angle1,
  double angle2) noexcept;
void arc_counterclockwise(const vector_2d& center, double radius,
  double angle1, double angle2) noexcept;
void ellipse(const experimental::io2d::ellipse& val) noexcept;
void ellipse(const circle& val) noexcept;
void cubic_curve_to(const vector_2d& pt0, const vector_2d& pt1,
  const vector_2d& pt2) noexcept;
void line_to(const vector_2d& pt) noexcept;
void move_to(const vector_2d& pt) noexcept;
void quadratic_curve_to(const vector_2d& pt0, const vector_2d& pt2)
  noexcept;
void rectangle(const experimental::io2d::rectangle& r) noexcept;
void rel_rectangle(const experimental::io2d::rectangle& r) noexcept;
void rel_ellipse(const experimental::io2d::ellipse& val) noexcept;
void rel_ellipse(const circle& val) noexcept;
void rel_cubic_curve_to(const vector_2d& dpt0, const vector_2d& dpt1,
const vector_2d& dpt2) noexcept;
void rel_line_to(const vector_2d& dpt) noexcept;
void rel_move_to(const vector_2d& dpt) noexcept;
void rel_quadratic_curve_to(const vector_2d& pt0, const vector_2d& pt2)
  noexcept;
void transform_matrix(const matrix_2d& m) noexcept;
void origin(const vector_2d& pt) 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;

      // 10.21.7, observers:
      experimental::io2d::rectangle path_extents() const 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);

    // 10.21.8, specialized algorithms:
    template <class Allocator>
    void swap(path_builder<Allocator>& lhs, path_builder<Allocator>& rhs)
      noexcept(noexcept(lhs.swap(rhs)));
  } } } }
```

10.21.2 path_builder container requirements [pathbuilder.containerrequirements]

1 This class shall be considered 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.

10.21.3 path_builder constructors, copy, and assignment [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.

10.21.4 `path_builder` capacity [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 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.

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

```
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.

```
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.

```
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.

10.21.5 path_builder modifiers [pathbuilder.modifiers]

```
void new_path() noexcept;
```

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

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

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

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

```
void arc_clockwise(const vector_2d& center, double radius, double angle1,
  double angle2) noexcept;
```

4 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::arc_clockwise(center, radius, angle1, angle2)` to the end of the path group.

```
void arc_counterclockwise(const vector_2d& center, double radius,
  double angle1, double angle2) noexcept;
```

5 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::arc_counterclockwise(center, radius, angle1, angle2)` to the end of the path group.

```
void ellipse(const experimental::io2d::ellipse& val) noexcept;
void ellipse(const circle& val) noexcept;
```

6 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::abs_ellipse(val)` to the end of the path group.

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

7 8*Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::abs_cubic_curve(pt0, pt1, pt2)` to the end of the path group.

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

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

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

10 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::abs_move(pt)` to the end of the path group.

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

11 12*Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::abs_quadratic_curve(pt0, pt1)` to the end of the path group.

```
void rectangle(const experimental::io2d::rectangle& r) noexcept;
```

13 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::abs_rectangle({ r })` to the end of the path group.

```
void rel_rectangle(const experimental::io2d::rectangle& r) noexcept;
```

14 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::rel_rectangle({ r })` to the end of the path group.

```
void rel_ellipse(const experimental::io2d::ellipse& val) noexcept;
void rel_ellipse(const circle& val) noexcept;
```

15 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::rel_ellipse({ val })` to the end of the path group.

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

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

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

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

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

18 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::rel_move(dpt)` to the end of the path group.

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

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

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

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

21 *Effects:* Adds a `path_data::path_data_types` object constructed from `(path_data::change_matrix(m)` to the end of the path group.

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

22 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::change_origin(pt)` 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);
```

23 *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.

24 *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();
```

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

26 *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.

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

```
void clear() noexcept;
```

28 *Postconditions:* 29 The current point shall not contain a value.

30 The transformation matrix shall have a value of `matrix_2d::init_identity{ }`.

31 The origin shall have a value of `vector_2d{ }`.

32 *Remarks:* The postconditions listed above are in addition to sequence container requirements for this function.

10.21.6 path_builder iterators [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_data_types` item in the path group.

2 *Remarks:* Changing a `path_data::path_data_types` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_data_types` object or of any subsequent `path_data::path_data_types` object in the path group shall result in undefined behavior when the path group is processed as described in 10.1 unless all of the violations are fixed prior to such processing.

```
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_data_types` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_data_types` object or of any subsequent `path_data::path_data_types` object in the path group shall result in undefined behavior when the path group is processed as described in 10.1 unless all of the violations are fixed prior to such processing.

```
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_data_types` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_data_types` object or of any subsequent `path_data::path_data_types` object in the path group shall result in undefined behavior when the path group is processed as described in 10.1 all of the violations are fixed prior to such processing.

```
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_data_types` object or otherwise modifying the path group in a way that violates the preconditions of that `path_data::path_data_types` object or of any subsequent `path_data::path_data_types` object in the path group shall result in undefined behavior when the path group is processed as described in 10.1 unless all of the violations are fixed prior to such processing.

10.21.7 path_builder observers [pathbuilder.observers]

```
experimental::io2d::rectangle path_extents() const noexcept;
```

1 *Returns:* A `rectangle` object which contains the extents of the path segments, including degenerate path segments, in the path group when it is processed as described in 10.1. [*Note:* By using path segments, this description intentionally excludes points established by `move_to` and `rel_move_to` operations from the extents value except where those points are subsequently used in defining a path segment. — *end note*]

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

2 *Returns:* If the current point contains a value, `true`, otherwise `false`.

```
vector_2d current_point() const noexcept;
```

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

4 *Returns:* The value of the current point.

```
matrix_2d transform_matrix() const noexcept;
```

5 *Returns:* The value of the transformation matrix.

```
vector_2d origin() const noexcept;
```

6 *Returns:* The value of the origin.

10.21.8 `path_builder` specialized algorithms [pathbuilder.special]

swap path_builder

```
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)`.

11 Brushes [brushes]

11.1 Overview of brushes [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 (12.16.3.2).

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` and `mask_props` classes both have a Wrap Mode, Filter and Matrix (12.3.1 and 12.4.1). Where necessary, these shall be referenced using those terms 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 (11.2).

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.

11.2 Gradient brushes [gradients]

11.2.1 Common properties of gradients [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.

11.2.2 Linear gradients [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<double>::epsilon()` is a *degenerate linear gradient*.

4 All attempts to sample from a a degenerate linear gradient return the color `bgra_color::transparent_black()`. The remainder of 11.2 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 0.0 corresponding to the begin point and a color stop offset value of 1.0 corresponding to the end point.

6 Color stop offset values in the range `(0.0,1.0)` linearly correspond to points on the line segment.

7 [*Example:* Given a linear gradient with a begin point of `vector_2d(0.0, 0.0)` and an end point of `vector_2d(10.0, 5.0)`, a color stop offset value of 0.6 would correspond to the point `vector_2d(6.0, 3.0)`. — *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.

11.2.3 Radial gradients [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<double>::epsilon()` and the difference between the radius of its start circle and the radius of its end circle is not greater than `numeric_limits<double>::epsilon()`; or,

(3.4) — its start circle has a radius of 0.0 and its end circle has a radius of 0.0.

4 All attempts to sample from a `brush` object created using a degenerate radial gradient return the color `bgra_color::transparent_black()`. The remainder of 11.2 is inapplicable to degenerate radial gradients.

5 A color stop offset of 0.0 corresponds to all points along the diameter of the start circle or to its center point if it has a radius value of 0.0.

6 A color stop offset of 1.0 corresponds to all points along the diameter of the end circle or to its center point if it has a radius value of 0.0.

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:

(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<double>::lowest(),numeric_limits<double>::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.

11.2.4 Sampling from gradients [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 `bgra_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 11.2.4 is inapplicable.

d) When no color stop has the offset value of `0.0`, then, defining n to be the offset value that is nearest to `0.0` 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 `bgra_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 `1.0`, then, defining n to be the offset value that is nearest to `1.0` 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 `bgra_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 `0.0` than a's offset value. [*Note:* This includes any color stops with an offset value of `0.0` provided that a's offset value is not `0.0`. — *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 `1.0` than b's offset value. [*Note:* This includes any color stops with an offset value of `1.0` provided that b's offset value is not `1.0`. — *end note*]

j) A color stop which has an offset value of `0.0` does not have an adjacent color stop to its left.

k) A color stop which has an offset value of `1.0` 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 `bgra_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 `0.0`, otherwise the color value of o shall be the same as-if the value of o was `1.0`.

(1.3) — If Wrap Mode is `wrap_mode::repeat`, then `1.0` 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 2.0 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 `1.0 - (o - 1.0)`, 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.

11.3 Enum class `wrap_mode` [wrapmode]

11.3.1 `wrap_mode` Summary [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 (12.16.3.2) 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 `bgra_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 `0.0`.

4 See Table 3 for the meaning of each `wrap_mode` enumerator.

11.3.2 `wrap_mode` Synopsis [wrapmode.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class wrap_mode {
    none,
    repeat,
    reflect,
    pad
  };
} } } }
```

11.3.3 `wrap_mode` Enumerators [wrapmode.enumerators]

Table 3 — `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 `bgra_color::transparent_black()`.
`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.

11.4 Enum class `filter` [filter]

11.4.1 `filter` Summary [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 4 for the meaning of each `filter` enumerator.

11.4.2 `filter` Synopsis [filter.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class filter {
    fast,
    good,
    best,
    nearest,
    bilinear
  };
} } } }
```

11.4.3 filter Enumerators [filter.enumerators]

Table 4 — 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.

11.5 Enum class brush_type [brushtype]

11.5.1 brush_type Summary [brushtype.summary]

1 The brush_type enum class denotes the type of a brush object.

2 See Table 5 for the meaning of each brush_type enumerator.

11.5.2 brush_type Synopsis [brushtype.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class brush_type {
    solid_color,
    surface,
    linear,
    radial
  };
} } } }
```

11.5.3 brush_type Enumerators [brushtype.enumerators]

Table 5 — brush_type enumerator meanings

Enumerator	Meaning
solid_color	The brush object is a solid color brush.

Table 5 — `brush_type` enumerator meanings (continued)

Enumerator	Meaning
`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.

11.6 Color stops [colorstops]

11.6.1 Class `color_stop` [colorstops.colorstop]

1 The class `color_stop` describes a color stop that is used by gradient brushes.

2 It has an offset of type `double` and a color of type `bgra_color`.

11.6.1.1 `color_stop` Synopsis [colorstops.colorstop.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class color_stop {
  public:
        // 11.6.1.2, construct:
    constexpr color_stop(double o, const bgra_color& c);

    // 11.6.1.3, modifiers:
    constexpr void offset(double val) noexcept;
        constexpr void color(const bgra_color& val) noexcept;

    // 11.6.1.4, observers:
        constexpr double offset() const noexcept;
        constexpr bgra_color color() const noexcept;
  };
} } } }
```

11.6.1.2 `color_stop` constructors [colorstops.colorstop.cons]

```
constexpr color_stop(double o, const bgra_color& c) noexcept;
```

1 *Effects:* Constructs a `color_stop` object.

2 The offset shall be set to the value of `o`.

3 The color shall be set to the value of `c`.

11.6.1.3 `color_stop` modifiers [colorstops.colorstop.modifiers]

```
constexpr void offset(double val) noexcept;
```

1 *Effects:* The offset shall be set to the value of `val`.

```
constexpr void color(double val) noexcept;
```

2 *Effects:* The color shall be set to the value of `val`.

11.6.1.4 `color_stop` observers [colorstops.colorstop.observers]

```
constexpr double offset() const noexcept;
```

1 *Returns:* The value of the offset.

```
constexpr bgra_color color() const noexcept;
```

2 *Returns:* The value of the color.

11.7 Class brush [brush]

11.7.1 brush Description [brush.intro]

1 The class `brush` describes an opaque wrapper for a graphics data graphics resource.

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 (12.16.3.2).

4 A `brush` object has a brush type of `brush_type`, which indicates which type of brush it is (Table 5).

5 As a result of technological limitations and considerations, a `brush` object's graphics data can have less precision than the data from which it was created.

6 [*Example:* Several graphics and rendering technologies that are currently widely used typically store individual color and alpha channel data as 8-bit unsigned normalized integer values while the `double` type that is used by the `bgra_color` class for individual color and alpha is often a 64-bit value. As such, it is possible for a loss of precision when transforming the 64-bit channel data of an `bgra_color` object to the 8-bit channel data that is commonly used internally in such graphics and rendering technologies. — *end example*]

11.7.2 brush synopsis [brush.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class brush {
  public:
    // 11.7.4, construct/copy/move/destroy:
    explicit brush(const bgra_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);

    // 11.7.5, observers:
    brush_type type() const noexcept;
  };
} } } }
```

11.7.3 Sampling from a brush object [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*.

11.7.3.1 Sampling from a solid color brush [brush.sampling.color]

1 When b is a solid color brush, then when sampling from b, the visual data returned shall always be the visual data equivalent bgra_color used to construct b, regardless of the point which is to be sampled and regardless of the return values of Wrap Mode, Filter, and Matrix.

11.7.3.2 Sampling from a linear gradient brush [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 Matrix using the requested point, from b, the visual data returned shall be as specified by 11.2.2 and 11.2.4.

11.7.3.3 Sampling from a radial gradient brush [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 Matrix using the requested point, from b, the visual data returned shall be as specified by 11.2.3 and 11.2.4.

11.7.3.4 Sampling from a surface brush [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 Matrix using the requested point, from b, the visual data returned shall be from the point pt in the graphics data of the brush, taking into account the values of Wrap Mode and Filter.

11.7.4 brush constructors and assignment operators [brush.cons]

```
explicit brush(const bgra_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 shall be created from the value of c. The visual data format of the graphics data shall be as-if it is that specified by format::argb.

4 *Remarks:* Sampling from this produces the results specified in 11.7.3.1.

```
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 color stop collection containing the values in the range [first,last).

6 The brush's brush type is brush_type::linear.

7 *Remarks:* Sampling from this brush produces the results specified in 11.7.3.2.

```
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 a color stop collection containing the color_stop objects in il.

9 The brush's brush type is brush_type::linear.

10 *Remarks:* Sampling from this brush produces the results specified in 11.7.3.2.

```
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 color stop collection containing the values in the range `[first,last)`.

12 The brush's brush type is `brush_type::radial`.

13 *Remarks:* Sampling from this brush produces the results specified in 11.7.3.3.

```
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 a color stop collection containing the `color_stop` objects in `il`.

15 The brush's brush type is `brush_type::radial`.

16 *Remarks:* Sampling from this brush produces the results specified in 11.7.3.3.

```
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 underlying raster graphics data graphics resource of `img`.

21 *Remarks:* Sampling from this brush shall produce the results specified in 11.7.3.4.

11.7.5 `brush` observers [brush.observers]

```
brush_type type() const noexcept;
```

1 *Returns:* The brush's brush type.

12 Surfaces [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 (12.16.3).

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.

12.1 Class `surface_props` [surfaceprops]

12.1.1 `surface_props` summary [surfaceprops.summary]

1 The `surface_props` class provides general state information that is applicable to all rendering and composing operations (12.16.3).

2 It has a Surface Matrix of type `matrix_2d`, an Antialiasing Value of type `antialias`, and a Compositing Operator of type `compositing_op`.

12.1.2 `surface_props` synopsis [surfaceprops.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class surface_props {
  public:
    // 12.1.3, constructors:
    constexpr surface_props() noexcept;
    constexpr explicit surface_props(const matrix_2d& m,
      antialias a = antialias::good,
      compositing_op co = compositing_op::over) noexcept;

    // 12.1.4, modifiers:
    constexpr void antialiasing(antialias a) noexcept;
    constexpr void compositing(compositing_op co) noexcept;
    constexpr void surface_matrix(const matrix_2d& m) noexcept;

    // 12.1.5, observers:
    constexpr antialias antialiasing() const noexcept;
    constexpr compositing_op compositing() const noexcept;
    constexpr matrix_2d surface_matrix() const noexcept;
  };
}}}}
```

12.1.3 `surface_props` constructors [surfaceprops.cons]

```
constexpr surface_props() noexcept;
```

1 *Effects:* The value of Surface Matrix is its default-constructed value.

2 The value of Antialiasing Value is `antialias::good`.

3 The value of Compositing Operator is `compositing_op::over`.

```
constexpr explicit surface_props(const matrix_2d& m,
  antialias a = antialias::good,
  compositing_op co = compositing_op::over) noexcept;
```

4 *Requires:* `m.is_invertible() == true`.

5 *Effects:* The value of Surface Matrix is `m`.

6 The value of Antialiasing Value is `a`.

7 The value of Compositing Operator is `co`.

12.1.4 surface_props modifiers [surfaceprops.modifiers]

```
constexpr void antialiasing(antialias a) noexcept;
```

1 *Effects:* The value of Antialiasing Value is `a`.

```
constexpr void compositing(compositing_op co) noexcept;
```

2 *Effects:* The value of Compositing Operator is `co`.

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

3 *Requires:* `m.is_invertible() == true`.

4 *Effects:* The value of Surface Matrix is `m`.

12.1.5 surface_props observers [surfaceprops.observers]

```
constexpr antialias antialiasing() const noexcept;
```

1 *Returns:* The value of Antialiasing Value.

```
constexpr compositing_op compositing() const noexcept;
```

2 *Returns:* The value of Compositing Operator.

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

3 *Returns:* The value of Surface Matrix.

12.2 Class stroke_props [strokeprops]

12.2.1 stroke_props summary [strokeprops.summary]

1 The `stroke_props` class provides state information that is applicable to the Stroke rendering and composing operation (12.16.3 and 12.16.7).

2 It has a Line Width of type `double`, a Line Cap of type `line_cap`, a Line Join of type `line_join`, and a Miter Limit of type `double`.

12.2.2 stroke_props synopsis [strokeprops.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class stroke_props {
  public:
    // 12.2.3, constructors:
    constexpr stroke_props() noexcept;
    constexpr explicit stroke_props(double w,
      experimental::io2d::line_cap lc = experimental::io2d::line_cap::none,
      experimental::io2d::line_join lj = experimental::io2d::line_join::miter,
      double ml = 10.0) noexcept

    // 12.2.4, modifiers:
    constexpr void line_width(double 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(double ml) noexcept;

    // 12.2.5, observers:
    constexpr double line_width() const noexcept;
    constexpr experimental::io2d::line_cap line_cap() const noexcept;
    constexpr experimental::io2d::line_join line_join() const noexcept;
    constexpr double miter_limit() const noexcept;
    constexpr double max_miter_limit() const noexcept;
  };
}}}}
```

12.2.3 stroke_props constructors [strokeprops.cons]

`constexpr stroke_props() noexcept;`

1 *Effects:* The value of Line Width is `2.0`.

2 The value of Line Cap is `experimental::io2d::line_cap::none`.

3 The value of Line Join is `experimental::io2d::line_join::miter`.

4 The value of Miter Limit is `10.0`.

```
constexpr explicit stroke_props(double w,
  experimental::io2d::line_cap lc = experimental::io2d::line_cap::none,
  experimental::io2d::line_join lj = experimental::io2d::line_join::miter,
  double ml = 10.0) noexcept
```

5 *Requires:* `w >= 0.0`.

6 `ml >= 1.0 && ml <= max_miter_limit()`

7 *Effects:* The value of Line Width is `w`.

8 The value of Line Cap is `lc`.

9 The value of Line Join is `lj`.

10 The value of Miter Limit is `ml`.

12.2.4 stroke_props modifiers [strokeprops.modifiers]

`constexpr void line_width(double w) noexcept;`

1 *Requires:* `w >= 0.0`.

2 *Effects:* The value of Line Width is `w`.

`constexpr void line_cap(experimental::io2d::line_cap lc) noexcept;`

3 *Effects:* The value of Line Cap is `lc`.

`constexpr void line_join(experimental::io2d::line_join lj) noexcept;constexpr`

4 *Effects:* The value of Line Join is `lj`.

`constexpr void miter_limit(double ml) noexcept;`

5 *Requires:* `ml >= 1.0 && ml <= max_miter_limit`.

6 The value of Miter Limit if `ml`.

12.2.5 stroke_props observers [strokeprops.observers]

```
constexpr double line_width() const noexcept;
```

1 *Returns:* The value of Line Width.

```
constexpr experimental::io2d::line_cap line_cap() const noexcept;
```

2 *Returns:* The value of Line Cap.

```
constexpr experimental::io2d::line_join line_join() const noexcept;
```

3 *Returns:* The value of Line Join.

```
constexpr double miter_limit() const noexcept;
```

4 *Returns:* The value of Miter Limit.

```
constexpr double max_miter_limit() const noexcept;
```

5 *Returns:* The implementation-defined maximum allowable value of Miter Limit.

6 *Remarks:* It is possible for this value to be `numeric_limits<double>::infinity()`.

12.3 Class brush_props [brushprops]

12.3.1 brush_props summary [brushprops.summary]

1 The `brush_props` class provides general state information that is applicable to all rendering and composing operations (12.16.3).

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`.

12.3.2 brush_props synopsis [brushprops.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class brush_props {
  public:
    // 12.3.3, constructors:
    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;

    // 12.3.4, modifiers:
    constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
    constexpr void filter(experimental::io2d::filter fi) noexcept;
    constexpr void fill_rule(experimental::io2d::fill_rule fr) noexcept;
    constexpr void brush_matrix(const matrix_2d& m) noexcept;

    // 12.3.5, observers:
    constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
    constexpr experimental::io2d::filter filter() const noexcept;
    constexpr experimental::io2d::fill_rule fill_rule() const noexcept;
    constexpr matrix_2d brush_matrix() const noexcept;
  };
}}}}
```

12.3.3 brush_props constructors [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:* The value of Wrap Mode is `w`.

2 The value of Filter is `fi`.

3 The value of Fill Rule is `fr`.

4 The value of Brush Matrix is `m`.

12.3.4 `brush_props` modifiers [brushprops.modifiers]

```
constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
```

1 *Effects:* The value of Wrap Mode is `w`.

```
constexpr void filter(experimental::io2d::filter fi) noexcept;
```

2 *Effects:* The value of Filter is `fi`.

```
constexpr void fill_rule(experimental::io2d::fill_rule fr) noexcept;
```

3 *Effects:* The value of Fill Rule is `fr`.

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

4 *Requires:* `m.is_invertible() == true`.

5 *Effects:* The value of Brush Matrix is `m`.

12.3.5 `brush_props` observers [brushprops.observers]

```
constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
```

1 *Returns:* The value of Wrap Mode.

```
constexpr experimental::io2d::filter filter() const noexcept;
```

2 *Returns:* The value of Filter.

```
constexpr experimental::io2d::fill_rule fill_rule() const noexcept;
```

3 *Returns:* The value of Fill Rule.

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

4 *Returns:* The value of Brush Matrix.

12.4 Class `mask_props` [maskprops]

12.4.1 `mask_props` summary [maskprops.summary]

1 The `mask_props` class provides state information that is applicable to the Mask rendering and composing operation (12.16.3).

2 It has a Wrap Mode of type `wrap_mode`, a Filter of type `filter`, and a Mask Matrix of type `matrix_2d`.

12.4.2 mask_props synopsis [maskprops.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class mask_props {
  public:
    // 12.4.3, 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;

    // 12.4.4, 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;

    // 12.4.5, observers:
    constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
    constexpr experimental::io2d::filter filter() const noexcept;
    constexpr matrix_2d mask_matrix() const noexcept;
  };
}}}}
```

12.4.3 mask_props constructors [maskprops.cons]

```
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
```

Requires: `m.is_invertible() == true`.

1 *Effects:* The value of Wrap Mode is `w`.

2 The value of Filter is `fi`.

3 The value of Mask Matrix is `m`.

12.4.4 mask_props modifiers [maskprops.modifiers]

```
constexpr void wrap_mode(experimental::io2d::wrap_mode w) noexcept;
```

1 *Effects:* The value of Wrap Mode is `w`.

```
constexpr void filter(experimental::io2d::filter fi) noexcept;
```

2 *Effects:* The value of Filter is `fi`.

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

3 *Requires:* `m.is_invertible() == true`.

4 *Effects:* The value of Mask Matrix is `m`.

12.4.5 mask_props observers [maskprops.observers]

```
constexpr experimental::io2d::wrap_mode wrap_mode() const noexcept;
```

1 *Returns:* The value of Wrap Mode.

```
constexpr experimental::io2d::filter filter() const noexcept;
```

2 *Returns:* The value of Filter.

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

3 *Returns:* The value of Mask Matrix.

12.5 Class clip_props [clipprops]

12.5.1 clip_props summary [clipprops.summary]

1 The `clip_props` class provides general state information that is applicable to all rendering and composing operations (12.16.3).

2 It has a Clip Area of type `path_group` and a Fill Rule of type `fill_rule`.

12.5.2 clip_props synopsis [clipprops.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class clip_props {
  public:
    // 12.5.3, constructors:
    clip_props() noexcept;
    template <class Allocator>
    explicit clip_props(const path_builder<Allocator>& pb,
      experimental::io2d::fill_rule = experimental::io2d::fill_rule::winding);
    explicit clip_props(const path_group& pg, experimental::io2d::fill_rule =
      experimental::io2d::fill_rule::winding) noexcept;

    // 12.5.4, 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;

      // 12.5.5, observers:
      path_group clip() const noexcept;
      experimental::io2d::fill_rule fill_rule() const noexcept;
    };
  }}}}
```

12.5.3 clip_props constructors [clipprops.cons]

`clip_props() noexcept;`

1 *Effects:* The value of Clip Area is its default-constructed value.

2 The value of Fill Rule is `experimental::io2d::fill_rule::winding`.

12.5.4 clip_props modifiers [clipprops.modifiers]

```
template <class Allocator>
void clip(const path_builder<Allocator>& pb);
void clip(const path_group& pg) noexcept;
```

1 *Effects:* The value of Clip Area is:

(1.1) — `path_group{pb}`; or

(1.2) — `pg`.

`void fill_rule(experimental::io2d::fill_rule fr) noexcept;`

2 *Effects:* The value of Fill Rule is `fr`.

12.5.5 clip_props observers [clipprops.observers]

`path_group clip() const noexcept;`

1 *Returns:* The value of Clip Area.

`experimental::io2d::fill_rule fill_rule() const noexcept;`

2 *Returns:* The value of Fill Rule.

12.6 Enum class antialias [antialias]

12.6.1 antialias Summary [antialias.summary]

1 The antialias enum class specifies the type of anti-aliasing that the rendering system shall use for rendering text. See Table 6 for the meaning of each `antialias` enumerator.

12.6.2 antialias Synopsis [antialias.synopsis]

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

12.6.3 antialias Enumerators [antialias.enumerators]

Table 6 — `antialias` enumerator meanings

Enumerator	Meaning
`default_antialias`	The meaning of this value is implementation-defined.
`none`	No anti-aliasing.
`gray`	Monochromatic anti-aliasing. [*Note:* When rendering black text on a white background, this would produce gray-scale
`subpixel`	Anti-aliasing that breaks pixels into their constituent color channels and manipulates those color channels individually. The meaning of this value for any rendering operation other than `surface::show_text`, `surface::show_glyphs`, and `surface::show_text_glyphs` is implementation-defined.
`fast`	The meaning of this value is implementation-defined. Implementations shall enable some form of anti-aliasing when this option is selected. [*Note:* By choosing this value, the user is hinting that faster anti-aliasing is preferable to better anti-aliasing. — *end note*]
`good`	The meaning of this value is implementation-defined. Implementations shall enable some form of anti-aliasing when this option is selected. [*Note:* By choosing this value, the user is hinting that sacrificing some performance to obtain better anti-aliasing is acceptable but that performance is still a concern.
`best`	The meaning of this value is implementation-defined. Implementations shall enable some form of text anti-aliasing when this option is selected. [*Note:* By choosing this value, the user is hinting that better anti-aliasing is more important than performance.

12.7 Enum class fill_rule [fillrule]

12.7.1 fill_rule Summary [fillrule.summary]

1 The `fill_rule` enum class determines how the Filling operation (12.16.6) 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 7 for the meaning of each `fill_rule` enumerator.

12.7.2 fill_rule Synopsis [fillrule.synopsis]

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

12.7.3 `fill_rule` Enumerators [fillrule.enumerators]

Table 7 — `fill_rule` enumerator meanings

Enumerator	Meaning
`winding`	If the Fill Rule (12.3.1) 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*]

12.8 Enum class `line_cap` [linecap]

12.8.1 `line_cap` Summary [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 8 for the meaning of each `line_cap` enumerator.

12.8.2 `line_cap` Synopsis [linecap.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class line_cap {
    none,
    round,
    square
  };
} } } }
```

12.8.3 `line_cap` Enumerators [linecap.enumerators]

Table 8 — `line_cap` enumerator meanings

Enumerator	Meaning
`none`	The line has no cap. It terminates exactly at the end point.

Table 8 — `line_cap` enumerator meanings (continued)

Enumerator	Meaning
`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.

12.9 Enum class `line_join` [linejoin]

12.9.1 `line_join` Summary [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 9 for the meaning of each enumerator.

12.9.2 `line_join` Synopsis [linejoin.synopsis]

```
namespace std { namespace experimental { namespace drawing { inline namespace
v1 {
  enum class line_join {
    miter,
    round,
    bevel
  };
} } } }
```

12.9.3 `line_join` Enumerators [linejoin.enumerators]

Table 9 — `line_join` enumerator meanings

Enumerator	Meaning
`miter`	Joins will be mitered or beveled, depending on the Miter Limit (12.2.1).
`round`	Joins will be rounded, with the center of the circle being the join point.
`bevel`	Joins will be beveled, with the join cut off at half the line width from the join point. Implementations may vary the cut off distance by an amount that is less than one pixel at each join for aesthetic or technical reasons.

12.10 Enum class `compositing_op` [compositingop]

12.10.1 `compositing_op` Summary [compositingop.summary]

1 The `compositing_op` enum class specifies composition algorithms. See Table 10, Table 11 and Table 12 for the meaning of each `compositing_op` enumerator.

12.10.2 `compositing_op` Synopsis [compositingop.synopsis]

```
namespace std { namespace experimental { namespace drawing { inline namespace
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
  };
} } } }
```

12.10.3 `compositing_op` Enumerators [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 10 — `compositing_op` basic enumerator meanings

Enumerator	**Bound**	**Color**	**Alpha**
`clear`	Yes	$Rc = 0$	$Ra = 0$
`source`	Yes	$Rc = Sc$	$Ra = Sa$
`over`	N/A	$Rc = \dfrac{(Sca + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da \times (1 - Sa)$
`in`	No	$Rc = Sc$	$Ra = Sa \times Da$

Table 10 — `compositing_op` basic enumerator meanings (continued)

Enumerator	Bound	Color	Alpha
`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	$Rc = \dfrac{(Sca \times (1 - Da) + Dca)}{Ra}$	$Ra = (1 - Da) \times Sa + Da$
`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	$Rc = \dfrac{(Sca \times (1 - Da) + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da - 2 \times Sa \times Da$
`add`	N/A	$Rc = \dfrac{(Sca + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$
`saturate`	N/A	$Rc = \dfrac{(min(Sa, 1 - Da) \times Sc + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$

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, 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 11 and Table 12 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 11 — `compositing_op` blend enumerator meanings

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

Table 11 — `compositing_op` blend enumerator meanings (continued)

Enumerator	**Color**
`overlay`	$if(Dc \leq 0.5)\ \{$ $f(Sc, Dc) = 2 \times Sc \times Dc$ $\}$ $else\ \{$ $f(Sc, Dc) =$ $1 - 2 \times (1 - Sc) \times$ $(1 - Dc)$ $\}$ [*Note:* The difference between this enumerator and `hard_light` is that this tests the destination color (Dc) whereas `hard_light` tests the source color (Sc). — *end note*]
`darken`	$f(Sc, Dc) = min(Sc, Dc)$
`lighten`	$f(Sc, Dc) = max(Sc, Dc)$
`color_dodge`	$if(Dc < 1)\ \{$ $f(Sc, Dc) = min(1, \frac{Dc}{(1 - Sc)})$ $\}$ $else\ \{$ $f(Sc, Dc) = 1\}$
`color_burn`	$if\ (Dc > 0)\ \{$ $f(Sc, Dc) = 1 - min(1, \frac{1 - Dc}{Sc})$ $\}$ $else\ \{$ $f(Sc, Dc) = 0$ $\}$
`hard_light`	$if\ (Sc \leq 0.5)\ \{$ $f(Sc, Dc) = 2 \times Sc \times Dc$ $\}$ $else\ \{$ $f(Sc, Dc) =$ $1 - 2 \times (1 - Sc) \times$ $(1 - Dc)$ $\}$ [*Note:* The difference between this enumerator and `overlay` is that this tests the source color (Sc) whereas `overlay` tests the destination color (Dc). — *end note*]

Table 11 — `compositing_op` blend enumerator meanings (continued)

Enumerator	Color

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

`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 12 — `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`	$f(Sc, Dc) = set\_lum(Dc,\ lum(Sc))$

12.11 Enum class `format` [format]

12.11.1 `format` Summary [format.summary]

1 The `format` enum class indicates a visual data format. See Table 13 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*]

12.11.2 `format` Synopsis [format.synopsis]

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

12.11.3 `format` Enumerators [format.enumerators]

Table 13 — `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.
`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.

12.12 Enum class `scaling` [scaling]

12.12.1 `scaling` Summary [scaling.summary]

1 The scaling enum class specifies the type of scaling a `display_surface` will use when the size of its Display Buffer (12.18.1) differs from the size of its Back Buffer (12.18.1).

2 See Table 14 for the meaning of each `scaling` enumerator.

12.12.2 `scaling` Synopsis [scaling.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class scaling {
    letterbox,
    uniform,
    fill_uniform,
    fill_exact,
    none
  };
} } } }
```

12.12.3 `scaling` Enumerators [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 (12.18.1) 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 $1.\overline{3}$.

The Display Buffer (12.18.1) of `ds` is 1280x720, giving it an aspect ratio of $1.\overline{7}$.

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 14 — `scaling` enumerator meanings

Enumerator	Meaning
`letterbox`	Fill the Display Buffer with the Letterbox Brush (12.18.4) 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 14 — `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. — *end note*]

Table 14 — `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*]

12.13 Enum class `refresh_rate` [refreshrate]

12.13.1 `refresh_rate` Summary [refreshrate.summary]

1 The `refresh_rate` enum class describes when the Draw Callback (Table 21) of a `display_surface` object shall be called. See Table 15 for the meaning of each enumerator.

12.13.2 `refresh_rate` Synopsis [refreshrate.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class refresh_rate {
    as_needed,
    as_fast_as_possible,
    fixed
  };
} } } }
```

12.13.3 `refresh_rate` Enumerators [refreshrate.enumerators]

Table 15 — `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.

Table 15 — `refresh_rate` value meanings (continued)

Enumerator	Meaning
`fixed`	The Draw Callback shall be called as frequently as needed to maintain the Desired Frame Rate (Table 21) 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 `20.0`, 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*]

12.14 Enum class `image_file_format` [imagefileformat]

12.14.1 `image_file_format` Summary [imagefileformat.summary]

1 The `image_file_format` enum class specifies the data format that an `image_surface` object shall be constructed from or shall be saved to.

2 This allows data in a format that is required to be supported to be read or written regardless of its extension.

12.14.2 `image_file_format` Synopsis [imagefileformat.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class image_file_format {
    png,
    jpg
  };
} } } }
```

12.14.3 `image_file_format` Enumerators [imagefileformat.enumerators]

Table 16 — `imagefileformat` enumerator meanings

Enumerator	Meaning
`png`	The data is in the PNG format.
`jpg`	The data is in the JPEG format.

12.15 Class `device` [device]

12.15.1 `device` synopsis [device.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class device {
  public:
    // See 4.3
    typedef implementation-defined native_handle_type; // exposition only
    native_handle_type native_handle() const noexcept; // exposition only

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

    // 12.15.3, modifiers:
    void flush() noexcept;
    void lock();
    void lock(error_code& ec) noexcept;
    void unlock();
    void unlock(error_code& ec) noexcept;
  };
} } } }
```

12.15.2 `device` Description [device.intro]

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

2 A `device` object is obtained from a `surface` or `surface`-derived object.

12.15.3 `device` modifiers [device.modifiers]

`void flush() noexcept;`

1 *Effects:* The user shall be able to manipulate the underlying rendering and presentation technologies used by the implementation without introducing a race condition.

2 *Postconditions:* Any pending device operations shall be executed, batched, or otherwise committed to the underlying rendering and presentation technologies.

3 Saved device state, if any, shall be restored.

4 *Remarks:* This function exists primarily to allow the user to take control of the underlying rendering and presentation technologies using an implementation-provided native handle.

5 The implementation's responsibility is to ensure that the user can safely make changes to the underlying rendering and presentation technologies using a native handle after calling this function.

6 The implementation is not required to ensure that every last operation has fully completed so long as those operations which are not complete do not prevent safe use of the underlying rendering and presentation technologies.

7 If the underlying technologies internally batch operations in a way that allows them to receive and batch further commands without introducing race conditions, the implementation should return as soon as all pending operations have been submitted to the batch queue.

8 This function should not flush the surface to which the device is bound.

9 If the implementation does not provide a native handle to the underlying rendering and presentation technologies, this function shall have no observable behavior.

10 *Notes:* Users call this function because they wish to use a native handle to the underlying rendering and presentation technologies in order to do something not provided by this Technical Specification (e.g. render native UI controls). As such, the user needs to know that using the underlying rendering system outside of this library will not introduce any race conditions. This function, in combination with locking the device, exists to provide that surety.

```
void lock();
void lock(error_code& ec) noexcept;
```

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

12 *Throws:* As described in Error reporting (5).

13 *Error conditions:* `errc::resource_unavailable_try_again` if a lock cannot be obtained. [*Note:* One reason this error may occur is if a system limit on the maximum number of times a lock could be recursively acquired would be exceeded. — *end note*]

```
void unlock();
void unlock(error_code& ec) noexcept;
```

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

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

16 *Throws:* As described in Error reporting (5).

17 *Remarks:* This function shall not be called more times than `lock` has been called; no diagnostic is required.

12.16 Class `surface` [surface]

12.16.1 `surface` description [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 12.16.3 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.

12.16.2 surface synopsis [surface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class surface {
  public:
    surface() = delete;

    // 12.16.9, 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;
    shared_ptr<experimental::io2d::device> device();
    shared_ptr<experimental::io2d::device> device(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);

    // 12.16.10, render modifiers:
    void paint(const brush& b, const optional<brush_props>& bp = nullopt,
      const optional<surface_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<surface_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<surface_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<surface_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<surface_props>& rp = nullopt,
      const optional<clip_props>& cl = nullopt);
    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<surface_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<surface_props>&rp = nullopt,
      const optional<clip_props>& cl = nullopt);
  };
} } } }
```

12.16.3 Rendering and composing [surface.rendering]

12.16.3.1 Operations [surface.rendering.ops]

1 The `surface` class provides four fundamental rendering and composing operations:

Table 17 — `surface` rendering and composing operations

Operation	Function(s)
Painting	`surface::paint`
Filling	`surface::fill`
Stroking	`surface::stroke`
Masking	`surface::mask`

12.16.3.2 Rendering and composing brushes [surface.rendering.brushes]

1 All rendering and composing operations use a Source Brush of type `brush`.

2 The Masking rendering and composing operation uses a Mask Brush of type `brush`.

12.16.3.3 Rendering and composing source path [surface.rendering.sourcepath]

1 In addition to brushes (12.16.3.2), all rendering and composing operation except for Painting use a Source Path of type `path_group`.

12.16.3.4 Common state data [surface.rendering.commonstate]

1 All rendering and composing operations use the following state data:

Table 18 — `surface` rendering and composing common state data

Name	Type
Brush Properties	`brush_props`
Surface Properties	`surface_props`
Clip Properties	`clip_props`

12.16.3.5 Specific state data [surface.rendering.specificstate]

1 In addition to the common state data (12.16.3.4), certain rendering and composing operations use state data that is specific each of them:

Table 19 — `surface` rendering and composing specific state data

Operation	Name	Type
Stroking	Stroke Properties	`stroke_props`
Masking	Mask Properties	`mask_props`

12.16.3.6 State data default values [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.

12.16.4 Standard coordinate spaces [surface.coordinatespaces]

1 There are four standard coordinate spaces relevant to the rendering and composing operations (12.16.3):

(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 (12.16.3.2). Its transformation matrix is the Brush Properties' Brush Matrix (12.3.1).

3 The *Mask Coordinate Space* is the standard coordinate space of the Mask Brush (12.16.3.2). Its transformation matrix is the Mask Properties' Mask Matrix (12.4.1).

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 (12.1.1).

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 20 — 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).`
Brush Coordinate Space	Surface Coordinate Space	`scsm.transform_point(bcsm.invert().transform_point(pt)).`

Table 20 — Point transformations (continued)

From	To	Transform
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).`

12.16.5 `surface` painting [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 (12.16.4) to the Brush Coordinate Space (Table 20), resulting in point *bp*.

3. Sample from point *bp* of the Source Brush (12.16.3.2), 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 Composition Operator (12.1.1), and modify the visual data of the underlying graphics data graphics resource at point *sp* to reflect the result produced by application of the Composition Operator.

12.16.6 `surface` filling [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 (12.5.1); if so, proceed with the remaining steps.

2. Transform *sp* from the Surface Coordinate Space (12.16.4) to the User Coordinate Space (Table 20), resulting in point *up*.

3. Using the Source Path (12.16.3.3) and the Fill Rule (12.3.1), 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 (12.16.4 and Table 20), resulting in point *bp*.

5. Sample from point *bp* of the Source Brush (12.16.3.2), 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 Composition Operator (12.1.1), and modify the visual data of the underlying graphics data graphics resource at point *sp* to reflect the result produced by application of the Composition Operator.

12.16.7 `surface` stroking [surface.stroking]

1 When a Stroking operation is initiated on a surface, the implementation shall carry out the Stroking operation for each path in the Source Path (12.16.3).

2 The following rules shall apply when a Stroking operation is carried out on a pathy:

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 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; 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 has its default value or if its `vector<double>` member is empty, the path segments shall be rendered as a continuous path.
5. If the Dash Pattern's `vector<double>` 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<double>` 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 `double` member, which specifies an offset value, is not `0.0`, 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. 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 (12.16.4) to the Surface Coordinate Space (12.16.4). [*Example:* If an open path consisting solely of a vertical line from `vector_2d(20.0, 20.0)` to `vector_2d(20.0, 60.0)` is to be composed to the underlying graphics data graphics resource, the Line Cap is `line_cap::none`, the Line Width is `12.0`, and the Transformation Matrix is `matrix_2d::init_scale(0.5, 1.0)`, then the line will end up being composed within the area `rectangle( { 7.0, 20.0 }, { 13.0, 60.0 } )` on the underlying graphics data graphics resource. The Transformation Matrix causes the center of the x axisof the line to move from `20.0` to `10.0` and then causes the horizontal width of the line to be reduced from `12.0` to `6.0`. — *end example*]

12.16.8 `surface` masking [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 (12.5.1); if so, proceed with the remaining steps.
2. Transform *sp* from the Surface Coordinate Space (12.16.4) to the Mask Coordinate Space (Table 20), 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 (12.16.3.2), 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 Composition Operator (12.1.1), multiply each channel of the result produced by application of the Composition 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.

12.16.9 surface state modifiers [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 (5).

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 *Notes:* 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 (12.16.3), 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.

```
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 (5).

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.

```
shared_ptr<experimental::io2d::device> device();
shared_ptr<experimental::io2d::device> device(error_code& ec) noexcept;
```

21 *Returns:* A shared pointer to the `device` object for this `surface`. If a `device` object does not already exist for this `surface`, a shared `device` object shall be allocated and returned.

22 *Throws:* As specified in Error reporting (5).

23 *Error conditions:* `errc::not_enough_memory` if a `device` object needs to be created and not enough memory exists to do so.

```
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);
```

24 *Effects:* Creates a `mapped_surface` object and calls `action` using it.

25 The `mapped_surface` object is created using `*this`, which allows direct manipulation of the underlying graphics data graphics resource.

26 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.

27 *Throws:* As specified in Error reporting (5).

28 *Remarks:* Whether changes are committed to the underlying graphics data graphics resource immediately or only when the `mapped_surface` object is destroyed is unspecified.

29 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.

30 *Error conditions:* The errors, if any, produced by this function are implementation-defined or are produced by the user-provided function passed via the `action` argument.

12.16.10 surface render modifiers [surface.modifiers.render]

```
void paint(const brush& b, const optional<brush_props>& bp = nullopt,
  const optional<surface_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

1 *Effects:* Performs the Painting rendering and composing operation as specified by 12.16.5.

2 The meanings of the parameters are specified by 12.16.3.

3 *Throws:* As specified in Error reporting (5).

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<surface_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<surface_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

5 *Effects:* Performs the Stroking rendering and composing operation as specified by 12.16.7.

6 The meanings of the parameters are specified by 12.16.3.

7 *Throws:* As specified in Error reporting (5).

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<surface_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<surface_props>& rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

9 *Effects:* Performs the Filling rendering and composing operation as specified by 12.16.6.

10 The meanings of the parameters are specified by 12.16.3.

11 *Throws:* As specified in Error reporting (5).

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<surface_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<surface_props>&rp = nullopt,
  const optional<clip_props>& cl = nullopt);
```

13 *Effects:* Performs the Masking rendering and composing operation as specified by 12.16.8.

14 The meanings of the parameters are specified by 12.16.3.

15 *Throws:* As specified in Error reporting (5).

16 *Error conditions:*

The errors, if any, produced by this function are implementation-defined.

12.17 Class image_surface [imagesurface]

12.17.1 image_surface summary [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*]

12.17.2 image_surface synopsis [imagesurface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class image_surface : public surface {
  public:
    // 12.17.3, 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);

    // 12.17.4, members:
    void save(filesystem::path p, image_file_format i);

    // 12.17.5, observers:
    experimental::io2d::format format() const noexcept;
    int width() const noexcept;
    int height() const noexcept;
  };
} } } }
```

12.17.3 image_surface constructors and assignment operators [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 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.

12 *Throws:* As specified in Error reporting [fs.err.report] in N4618.

13 *Error conditions:* Any error that could result from trying to access `f`, open `f` for reading, or reading data from `f`.

14 Other errors, if any, produced by this function are implementation-defined.

12.17.4 image_surface members [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 (12.16.3) shall be 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 *Throws:* As specified in Error reporting [fs.err.report] in N4618.

6 *Error conditions:* Any error that could result from trying to create `f`, access `f`, or write data to `f`.

7 Other errors, if any, produced by this function are implementation-defined.

12.17.5 `image_surface` observers [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`.

12.18 Class `display_surface` [displaysurface]

12.18.1 `display_surface` Description [displaysurface.intro]

1 The class `display_surface` derives from the `surface` class and provides an interface to a raster graphics data graphics resource called the `Back Buffer` and to a second raster graphics data graphics resource 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 21) 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*]

12.18.2 `display_surface` synopsis [displaysurface.synopsis]

```
  namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class display_surface : public surface {
    public:
```

```
// 12.18.5, 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, double fps = 30.0);
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, double fps = 30.0)
  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, double fps = 30.0);
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, double fps = 30.0)
  noexcept;

~display_surface();

// 12.18.6, 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(double fps) noexcept;
      void redraw_required() noexcept;
      int begin_show();
      void end_show();

      // 12.18.7, 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;
      double desired_frame_rate() const noexcept;
      double elapsed_draw_time() const noexcept;
    };
  } } } }
```

12.18.3 display_surface miscellaneou behavior [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.

12.18.4 display_surface state [displaysurface.state]

1 Table 21 specifies the name, type, function, and default value for each item of a display surface's observable state.

Table 21 — 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 14)	brush{ { bgra_color::black() } }
Letterbox Brush Props	brush_props	This is the brush properties for the Letterbox Brush	brush_props{ }

Table 21 — Display surface observable state (continued)

Name	Type	Function	Default value
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	`antialias::default_-antialias`
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 21 — 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 21 — 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 21 — 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 21 — 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< experimental:: io2d:: 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 14). 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 21 — 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	`double`	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 15)	

12.18.5 `display_surface` constructors and assignment operators [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, double fps = 30.0);
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, double fps = 30.0)
  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 (5).

12 *Error conditions:* `io2d::device_error` if successful creation of the `display_surface` object would exceed the maximum number of simultaneous valid `display_surface` objects that the implementation supports.

13 Other errors, if any, produced by this function are implementation-defined.

```
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, double fps = 30.0);
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, double fps = 30.0)
  noexcept;
```

14 *Requires:* `preferredWidth > 0`.

15 `preferredHeight > 0`.

16 `preferredDisplayWidth > 0`.

17 `preferredDisplayHeight > 0`.

18 `preferredFormat != experimental::io2d::format::invalid`.

19 *Effects:* Constructs an object of type `display_surface`.

20 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.

21 The preferredFormat parameter specifies the preferred pixel format value for Draw Format.

22 The value of Scaling Type shall be the value of `scl`.

23 The value of Refresh Rate shall be the value of `rr`.

24 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.

25 All other observable state data shall have their default values.

26 *Throws:* As specified in Error reporting (5).

27 *Error conditions:* `io2d::device_error` if successful creation of the `display_surface` object would exceed the maximum number of simultaneous valid `display_surface` objects that the implementation supports.

28 Other errors, if any, produced by this function are implementation-defined.

12.18.6 `display_surface` modifiers [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 21 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 (5).

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 21 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 (5).

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 21 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 (5).

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 21 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 (5).

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 21 with `w` treated as being the preferred Draw Width and Draw Height is set as specified by Table 21 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 (5).

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 21 with `w` treated as being the preferred Display Height and Display Height is set as specified by Table 21 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 (5).

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 (5).

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(double 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 (5).

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 con*72*ditions:* `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.

12.18.7 `display_surface` observers [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 (5).

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.

```
double desired_framerate() const noexcept;
```

14 *Returns:* The value of Desired Framerate.

```
double 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 `0.0`.

12.19 Class mapped_surface [mappedsurface]

12.19.1 mapped_surface synopsis [mappedsurface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class mapped_surface {
  public:
    // 12.19.3, construct/copy/move/destroy:
    mapped_surface() = delete;
    mapped_surface(const mapped_surface&) = delete;
    mapped_surface& operator=(const mapped_surface&) = delete;
    mapped_surface(mapped_surface&& other) = delete;
    mapped_surface& operator=(mapped_surface&& other) = delete;
    ~mapped_surface();

    // 12.19.4, 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;

    // 12.19.5, 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;
  };
} } } }
```

12.19.2 mapped_surface Description [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 can only be created by the surface::map function. It cannot be copied or moved.

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 (12.11).

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.

12.19.3 `mapped_surface` constructors and assignment operators [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.

12.19.4 `mapped_surface` modifiers [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 (5).

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(bgra_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 *Throws:* As specified in Error reporting (5).

6 *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.

7 *Error conditions:* `io2d_error::null_pointer` if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

12.19.5 `mapped_surface` observers [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(bgra_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 *Throws:* As specified in Error reporting (5).

3 *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.

4 *Error conditions:* `io2d_error::null_pointer` if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

```
experimental::io2d::format format() const noexcept;
```

5 *Returns:* The pixel format of the mapped surface.

6 *Remarks:* If the mapped surface is invalid, this function shall return `experimental::io2d::format::invalid`.

```
int width() const noexcept;
```

7 *Returns:* The number of pixels per horizontal line of 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 height() const noexcept;
```

9 *Returns:* The number of horizontal lines of pixels in the mapped surface.

10 *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;
```

11 *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*]

12 *Remarks:* This function shall return the value `0` if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

13 Standalone functions [io2d.standalone]

13.1 Standalone functions synopsis [io2d.standalone.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace 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, double fps = 30.0);
  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, double fps = 30.0) 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, double fps = 30.0);
  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, double fps = 30.0) 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;
} } } } // namespaces std::experimental::io2d::v1
```

13.2 `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`.

13.3 `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, double fps = 30.0);
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, double fps = 30.0)
  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, double fps = 30.0);
```

```
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, double fps = 30.0)
  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 (5).

3 *Error conditions:* The errors, if any, produced by this function are the same as the errors for the equivalent `display_surface` constructor (12.18.5).

13.4 `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 (5).

3 *Error conditions:* The errors, if any, produced by this function are the same as the errors for the equivalent `display_surface` constructor (12.17.3).

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

2D graphics
  synopsis, 12–17

additive color, 4
aliasing, 3
alpha, 3
anti-aliasing, 3
artifact, 3
aspect ratio, 4

Bézier curve, 4
  cubic, 4
  quadratic, 4

C
  Unicode TR, 2
channel, 3
closed path, 7
color
  transparent black, 121
color model, 4
  RGB, 4
  RGBA, 4
color space, 4
  sRGB, 4
color stop, 6
compose, 6
composing operation, 6
composition algorithm, 6
control point, 6
CSS Colors Specification, 2
current point, 7

definitions, 3–8
degenerate path, 7
degenerate path segment, 6

filter, 4
final path segment, 7
format
  JPEG, 2
  PNG, 2

graphics data, 4
  raster, 4
graphics resource, 5
  graphics data graphics resource, 5
graphics state data, 5
graphics subsystem, 5

initial path segment, 6

last-move-to point, 7

normalize, 5

open path, 7

path, 7
path group, 7
path instruction, 7
path item, 7
path segment, 6
pixel, 3
pixmap, 5
point, 5
premultiplied format, 5

references
  normative, 2
render, 6
rendering and composing operation, 6
rendering operation, 6

sample, 6
scope, 1
standard coordinate space, 3

visual data, 3
visual data element, 3
visual data format, 3

Index of library names

- abs_cubic_curve, 63
  - constructor, 63
  - control_point_1, 64
  - control_point_2, 64
  - end_point, 64
- abs_ellipse, 64, 79
  - center, 65
  - constructor, 64, 65
  - x_axis, 65
  - y_axis, 65
- abs_line, 66
  - constructor, 66
  - to, 66
- abs_move, 66
  - constructor, 67
  - to, 67
- abs_quadratic_curve, 67
  - constructor, 68
  - control_point, 68
  - end_point, 68
- abs_rectangle, 68
  - bottom_right, 70
  - constructor, 69
  - height, 70
  - top_left, 70
  - width, 70
  - x, 70
  - y, 70
- angular_direction
  - vector_2d, 41
- antialias, 115
- arc_clockwise, 71
  - angle_1, 72
  - angle_2, 72
  - center, 72
  - circle, 72
  - constructor, 71, 72
  - path_builder, 91
  - radius, 72
- arc_counterclockwise, 73
  - angle_1, 74
  - angle_2, 74
  - center, 74
  - circle, 74
  - constructor, 73, 74
  - radius, 74
- auto_clear
  - display_surface, 158, 160

- begin
  - path_builder, 94
- bgra_color, 22
  - a, 26, 27
  - alice_blue, 27
  - antique_white, 27
  - aqua, 27
  - aquamarine, 27
  - azure, 27
  - b, 26, 27
  - beige, 27
  - bisque, 27
  - black, 27
  - blanched_almond, 27
  - blue, 27
  - blue_violet, 28
  - brown, 28
  - burly_wood, 28
  - cadet_blue, 28
  - chartreuse, 28
  - chocolate, 28
  - constructor, 26
  - coral, 28
  - cornflower_blue, 28
  - cornsilk, 28
  - crimson, 28
  - cyan, 28
  - dark_blue, 28
  - dark_cyan, 29
  - dark_goldenrod, 29
  - dark_gray, 29
  - dark_green, 29
  - dark_grey, 29
  - dark_khaki, 29
  - dark_magenta, 29
  - dark_olive_green, 29
  - dark_orange, 29
  - dark_orchid, 29
  - dark_red, 29
  - dark_salmon, 29
  - dark_sea_green, 30
  - dark_slate_blue, 30

dark_slate_gray, 30
dark_slate_grey, 30
dark_turquoise, 30
dark_violet, 30
deep_pink, 30
deep_sky_blue, 30
dim_gray, 30
dim_grey, 30
dodger_blue, 30
firebrick, 30
floral_white, 31
forest_green, 31
fuchsia, 31
g, 26
gainsboro, 31
ghost_white, 31
gold, 31
goldenrod, 31
gray, 31
green, 31
green_yellow, 31
grey, 31
honeydew, 31
hot_pink, 32
indian_red, 32
indigo, 32
ivory, 32
khaki, 32
lavender, 32
lavender_blush, 32
lawn_green, 32
lemon_chiffon, 32
light_blue, 32
light_coral, 32
light_cyan, 32
light_goldenrod_yellow, 33
light_gray, 33
light_green, 33
light_grey, 33
light_pink, 33
light_salmon, 33
light_sea_green, 33
light_sky_blue, 33
light_slate_gray, 33
light_slate_grey, 33
light_steel_blue, 33
light_yellow, 33
lime, 34
lime_green, 34
linen, 34
magenta, 34
maroon, 34
medium_aquamarine, 34
medium_blue, 34
medium_orchid, 34
medium_purple, 34
medium_sea_green, 34
medium_slate_blue, 34
medium_spring_green, 34
medium_turquoise, 35
medium_violet_red, 35
midnight_blue, 35
mint_cream, 35
misty_rose, 35
moccasin, 35
navajo_white, 35
navy, 35
old_lace, 35
olive, 35
olive_drab, 35
operator==, 39
orange, 35
orange_red, 36
orchid, 36
pale_goldenrod, 36
pale_green, 36
pale_turquoise, 36
pale_violet_red, 36
papaya_whip, 36
peach_puff, 36
peru, 36
pink, 36
plum, 36
powder_blue, 36
purple, 37
r, 26
red, 37
rosy_brown, 37
royal_blue, 37
saddle_brown, 37
salmon, 37
sandy_brown, 37
sea_green, 37
sea_shell, 37
sienna, 37
silver, 37
sky_blue, 37
slate_blue, 38
slate_gray, 38
slate_grey, 38
snow, 38
spring_green, 38

steel_blue, 38
tan, 38
teal, 38
thistle, 38
tomato, 38
transparent_black, 38
turquoise, 38
violet, 39
wheat, 39
white, 39
white_smoke, 39
yellow, 39
yellow_green, 39
brush, 104
constructor, 105, 106
type, 106
brush_props
brush_matrix, 112
constructor, 110
fill_rule, 112
filter, 112
wrap_mode, 112

capacity
path_builder, 90
cbegin
path_builder, 94
cend
path_builder, 94
change_matrix, 75
constructor, 75
matrix, 75
change_origin, 75
constructor, 76
origin, 76
circle, 44
center, 45
constructor, 45
radius, 45
clear
path_builder, 93
clip_props
clip, 115
constructor, 115
fill_rule, 115
close_path, 76
constructor, 77
to, 77
color
color_stop, 103
color_stop, 103
color, 103
constructor, 103
offset, 103
commit_changes
mapped_surface, 162
control_point_1
rel_cubic_curve, 79
control_point_2
rel_cubic_curve, 79
crbegin
path_builder, 94
crend
path_builder, 94
current_point
path_builder, 95

data
mapped_surface, 162, 163
desired_frame_rate
display_surface, 158
determinant
matrix_2d, 52
device, 132
flush, 132
lock, 133
surface, 140
unlock, 133
dimensions
display_surface, 156, 160
display_dimensions
display_surface, 157, 160
display_height
display_surface, 156, 160
display_surface, 144
auto_clear, 158, 160
begin_show, 158
constructor, 152, 153
desired_frame_rate, 158
dimensions, 156, 160
display_dimensions, 157, 160
display_height, 156, 160
display_width, 155, 160
draw_callback, 154
elapsed_draw_time, 160
end_show, 159
format, 160
height, 154, 160
letterbox_brush, 158, 160
redraw_required, 158
refresh_rate, 158
scaling, 158, 160

size_change_callback, 154
user_scaling_callback, 158, 160
width, 154, 160
display_width
display_surface, 155, 160
dot
vector_2d, 41
draw_callback
display_surface, 154

elapsed_draw_time
display_surface, 160
ellipse, 46
center, 47
constructor, 46
x_axis, 47
y_axis, 47
end
path_builder, 94
end_point
rel_cubic_curve, 79
equivalent
io2d_error_category, 20
erase
path_builder, 93
<experimental/io2d>, 12–17

fill
surface, 141
flush
device, 132
surface, 139
format
display_surface, 160
image_surface, 144
mapped_surface, 163
format_stride_for_width, 165

has_current_point
path_builder, 94
height
display_surface, 154, 160
image_surface, 144
mapped_surface, 163

image_file_format, 131
image_surface, 142
constructor, 143
format, 144
height, 144
save, 143
width, 144
init_identity
matrix_2d, 49
init_rotate
matrix_2d, 49
init_scale
matrix_2d, 49
init_shear_y
matrix_2d, 49
init_translate
matrix_2d, 49
invert
matrix_2d, 50
io2d_category, 21
io2d_error, 18
io2d_error_category, 20
equivalent, 20
message, 20
name, 20
is_finite
matrix_2d, 51
is_invertible
matrix_2d, 52

letterbox_brush
display_surface, 158, 160
literals
operator""ubyte, 39
operator""unorm, 39
lock
device, 133

m00
matrix_2d, 50, 51
m01
matrix_2d, 50, 51
m10
matrix_2d, 50, 51
m11
matrix_2d, 50, 51
m20
matrix_2d, 50, 51
m21
matrix_2d, 50, 51
magnitude
vector_2d, 41
magnitude_squared
vector_2d, 41
make_display_surface, 165
make_error_code, 21
make_error_condition, 21

make_image_surface, 166
map
    surface, 140
mapped_surface, 161
    commit_changes, 162
    data, 162, 163
    destructor, 162
    format, 163
    height, 163
    stride, 163
    width, 163
mark_dirty
    surface, 140
mask
    surface, 142
mask_props
    constructor, 113
    filter, 114
    mask_matrix, 114
    wrap_mode, 114
matrix_2d, 47
    constructor, 49
    determinant, 52
    init_identity, 49
    init_rotate, 49
    init_scale, 49
    init_shear_x, 49
    init_shear_y, 49
    init_translate, 49
    invert, 50
    is_finite, 51
    is_invertible, 52
    m00, 50, 51
    m01, 50, 51
    m10, 50, 51
    m11, 50, 51
    m20, 50, 51
    m21, 50, 51
    operator*, 52
    operator*=, 52
    operator==, 52
    rotate, 50
    scale, 50
    shear_x, 50
    shear_y, 50
    transform_point, 52
    translate, 50
message
    io2d_error_category, 20

name
    io2d_error_category, 20
new_path, 77

offset
    color_stop, 103
operator""ubyte
    literals, 39
operator""unorm
    literals, 39
operator*
    matrix_2d, 52
    vector_2d, 42
operator*=
    matrix_2d, 52
    vector_2d, 42
operator+
    vector_2d, 42
operator+=
    vector_2d, 42
operator-
    vector_2d, 42
operator-=
    vector_2d, 42
operator==
    matrix_2d, 52
    vector_2d, 42
origin
    change_origin, 76
    path_builder, 95

paint
    surface, 141
path_builder, 95
path_builder, 86
    arc_clockwise, 91
    arc_counterclockwise, 91
    begin, 94
    capacity, 90
    cbegin, 94
    cend, 94
    clear, 93
    close_path, 91
    constructor, 89, 90
    crbegin, 94
    crend, 94
    cubic_curve_to, 91
    current_point, 95
    ellipse, 91
    emplace_back, 93
    end, 94
    erase, 93

has_current_point, 94
insert, 93
line_to, 92
move_to, 92
new_path, 91
origin, 93, 95
path_extents, 94
pop_back, 93
push_back, 93
quadratic_curve_to, 92
rbegin, 94
rectangle, 92
rel_cubic_curve_to, 92
rel_ellipse, 92
rel_line_to, 92
rel_move_to, 92
rel_quadratic_curve_to, 92
rel_rectangle, 92
rend, 94
reserve, 90
resize, 91
shrink_to_fit, 90
swap, 90
transform_matrix, 92, 95

path_extents
- path_builder, 94

path_group, 85
- constructor, 86

pop_back
- path_builder, 93

rbegin
- path_builder, 94

rectangle, 42
- bottom_right, 44
- constructor, 43
- height, 44
- top_left, 44
- width, 44
- x, 44
- y, 44

red
- bgra_color, 37

redraw_required
- display_surface, 158

refresh_rate
- display_surface, 158

rel_cubic_curve, 77
- control_point_1, 78, 79
- control_point_2, 78, 79
- end_point, 78, 79

rel_cubic_curve_to
- constructor, 78

rel_ellipse
- center, 80
- constructor, 79, 80
- x_axis, 80
- y_axis, 80

rel_line, 80
- constructor, 81
- to, 81

rel_move, 81
- constructor, 82
- to, 82

rel_quadratic_curve, 82
- constructor, 83
- control_point, 83
- end_point, 83

rel_rectangle, 83
- bottom_right, 85
- constructor, 84
- height, 85
- top_left, 85
- width, 85
- x, 84, 85
- y, 84, 85

rend
- path_builder, 94

rosy_brown
- bgra_color, 37

rotate
- matrix_2d, 50

royal_blue
- bgra_color, 37

scale
- matrix_2d, 50

scaling
- display_surface, 158, 160

shear_x
- matrix_2d, 50

shear_y
- matrix_2d, 50

size_change_callback
- display_surface, 154

stride
- mapped_surface, 163

stroke
- surface, 141

stroke_props
- constructor, 109
- line_cap, 109, 110

line_join, 109, 110
line_width, 109, 110
max_miter_limit, 110
miter_limit, 109, 110
surface, 133
device, 140
fill, 141
flush, 139
map, 140
mark_dirty, 140
mask, 142
paint, 141
stroke, 141
surface_props
antialiasing, 108
compositing, 108
constructor, 107, 108
surface_matrix, 108
swap, 95

to_unit
vector_2d, 41
transform_matrix
path_builder, 95
transform_point
matrix_2d, 52
translate
matrix_2d, 50

unlock
device, 133
user_scaling_callback
display_surface, 158, 160

vector_2d, 40
angular_direction, 41
constructor, 41
dot, 41
magnitude, 41
magnitude_squared, 41
operator*, 42
operator*=, 42
operator+, 42
operator+=, 42
operator-, 42
operator-=, 42
operator==, 42
to_unit, 41
x, 41
y, 41

width
display_surface, 154, 160
image_surface, 144
mapped_surface, 163

x
vector_2d, 41

y
vector_2d, 41

Index of implementation-defined behavior

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

errc::argument_out_of_domain
    what_arg value, 10
errc::invalid_argument
    what_arg value, 10
io2d_error_category
    equivalent, 20, 21
numeric_limits<double>::is_iec559 evaluates to false, 9

antialias
    subpixel, 116
antialiasing
    best, 116
    default, 116
    fast, 116
    good, 116

Dash Pattern
    offset value, 138
display_surface
    constructor, 153
    dimensions, 157
    display_dimensions, 158
    display_height, 156
    display_width, 156
    height, 155
    letterbox_brush, 158
    maximum frame rate, 158
    minimum frame rate, 158
    show, 159
    show return value, 159
    unsupported Draw Format, 148
    width, 154

filter
    best, 102
    fast, 102
    good, 102

image_surface
    data, 143, 144
io2d_error
    device_error, 19
    invalid_status, 19
    null_pointer, 19
io2d_error_category
    message, 20

mapped_surface
    commit_changes, 162

other error codes
    what_arg value, 10
output device, 146

presence and meaning of native_handle_type and native_handle, 9

stroke_props
    max_miter_limit, 110
surface
    fill, 142
    map, 141
    mark_dirty, 140
    mask, 142
    paint, 141
    stroke, 141
surface::flush errors, 139

type of path_builder::const_iterator, 13, 87
type of path_builder::iterator, 13, 87
type of path_builder::size_type, 13, 87