Document Number: P0267R3
Date: 2017-02-06
Revises: P0267R2
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

List of Tables

List of Figures

1 General [io2d.general]

1.1 Scope [io2d.general.scope]

1 This Technical Specification specifies requirements for implementations of an interface that computer programs written in the C++ programming language may use to render and display 2D computer graphics.

1.2 Normative references [io2d.general.refs]

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

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

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

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

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

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

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

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

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

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

1.3 Terms and definitions [io2d.general.defns]

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

1.3.1 [io2d.general.defns.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

1.3.2 [io2d.general.defns.visualdata]

visual data

data representing color, transparency, or some combination thereof

1.3.3 [io2d.general.defs.channel]

channel

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

1.3.4 [io2d.general.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*]

1.3.5 [io2d.general.defns.visualdataelement]

visual data element

an item of visual data with a defined visual data format

1.3.6 [io2d.general.defns.alpha]

alpha

visual data representing transparency

1.3.7 [io2d.general.defns.pixel]

pixel

a discrete, rectangular visual data element

1.3.8 [io2d.general.defns.alias]

aliasing

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

1.3.9 [io2d.general.defns.artifact]

artifact

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

1.3.10 [io2d.general.defns.antialias]

anti-aliasing

the application of a function or algorithm while rendering 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*]

1.3.11 [io2d.general.defns.aspectratio]

aspect ratio

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

1.3.12 [io2d.general.defns.colormodel]

color model

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

1.3.13 [io2d.general.defns.additivecolor]

additive color

a color defined by the emissive intensity of its color channels

1.3.14 [io2d.general.defns.rgbcolormodel]

RGB color model

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

1.3.15 [io2d.general.defns.rgbacolormodel]

RGBA color model

the RGB color model with an alpha channel [*Note:* RGBA is not a proper color model; it is a convenient way to refer to the RGB color model to which an alpha channel has been added.

The interpretation of the alpha channel and its effect on the interpretation of the RGB color channels is intentionally not defined here because it is context-dependent. — *end note*]

1.3.16 [io2d.general.defns.colorspace]

color space

an unambiguous mapping of values to colorimetric colors [*Note:* The difference between a color model and a color space is often obscured, and sometimes the terms themselves are mistakenly used interchangeably.

A color model defines color mathematically without regard to how humans actually perceive color. Color models are useful for working with color computationally but, since they deal in ideal colors rather than perceived colors, they fail to provide the information necessary to allow for the uniform display of their colors on different output devices (e.g. LCD monitors, CRT TVs, and printers).

A color space, by contrast, maps unambiguous values to perceived 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. As such, the uniform display of the colors in a color space on different output devices is possible. — *end note*]

1.3.17 [io2d.general.defns.srgbcolorspace]

sRGB color space

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

1.3.18 [io2d.general.defns.cubicbezier]

cubic Bézier curve

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

1.3.19 [io2d.general.defns.quadraticbezier]

quadratic Bézier curve

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

1.3.20 [io2d.general.defns.filter]

filter

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

1.3.21 [io2d.general.defns.graphicsdata]

graphics data

<graphics data> visual data stored in an unspecified form

1.3.22 [io2d.general.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 $(0, 0)$

1.3.23 [io2d.general.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*]

1.3.24 **[io2d.general.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

1.3.25 **[io2d.general.defns.pixmap]**

pixmap

a raster graphics data graphics resource

1.3.26 **[io2d.general.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

1.3.27 **[io2d.general.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

1.3.28 **[io2d.general.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*]

1.3.29 **[io2d.general.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

1.3.30 **[io2d.general.defns.graphicssubsystem]**

graphics subsystem

collection of unspecified operating system and library functionality used to render and display 2D computer graphics

1.3.31 **[io2d.general.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*]

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

render

to transform a path group into graphics data in the manner specified by a set of graphics state data

1.3.33 **[io2d.general.defns.renderingoperation]**

rendering operation

an operation that performs rendering

1.3.34 **[io2d.general.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

1.3.35 **[io2d.general.defns.composingoperation]**

composing operation

an operation that performs composing

1.3.36 **[io2d.general.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

1.3.37 **[io2d.general.defns.renderingandcomposingop]**

rendering and composing operation

an operation that is either a composing operation or a rendering operation followed by a composing operation

1.3.38 **[io2d.general.defns.sample]**

sample

to use a filter to obtain the visual data for a given point from a graphics data graphics resource

1.3.39 **[io2d.general.defns.colorstop]**

color stop

a tuple composed of a floating point offset value in the range $[0, 1]$ and a color value

1.3.40 **[io2d.general.defns.pathsegment]**

path segment

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

1.3.41 **[io2d.general.defns.controlpoint]**

constrol point

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

1.3.42 **[io2d.general.defns.degeneratepathsegment]**

degenerate path segment

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

1.3.43 **[io2d.general.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*]

1.3.44 **[io2d.general.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*]

1.3.45 **[io2d.general.defns.pathinstruction]**

path instruction

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

1.3.46 **[io2d.general.defns.path]**

path

a collection of path instructions and path segments 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

1.3.47 **[io2d.general.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*]

1.3.48 **[io2d.general.defns.lastmovetopoint]**

last-move-to point

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

1.3.49 **[io2d.general.defns.pathgroup]**

path group

a collection of paths

1.3.50 **[io2d.general.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

1.3.51 **[io2d.general.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*]

1.3.52 **[io2d.general.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*]

2 Requirements [io2d.req]

2.1 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::`.

2.2 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 201606`

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*]

2.3 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*]

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

2.5 Exact width integer types [io2d.req.cstdintopttypes]

1 In order to implement this Technical Specification, the implementation shall provide the following optional integer types from the `<cstdint>` header file:

(1.1) — `uint8_t`

(1.2) — `uint16_t`

(1.3) — `uint32_t`

(1.4) — `uint64_t`

3 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.

4 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_operator;
  enum class format;
  enum class tiling;
  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 rgba_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 arc_clockwise;
  class arc_counterclockwise;
  class change_matrix;
  class change_origin;
  class close_path;
  class abs_cubic_curve;
  class abs_quadratic_curve;
  class abs_line;
  class abs_move;
  class new_path;
  class rel_cubic_curve;
  class rel_quadratic_curve;
  class rel_line;
  class rel_move;
  using path_data_types = typename variant<abs_move, rel_move,
    abs_line, rel_line, abs_cubic_curve, rel_cubic_curve,
    abs_quadratic_curve, rel_quadratic_curve, arc_clockwise,
    arc_counterclockwise, new_path, close_path, change_matrix, change_origin>;
};
class path;
template <Allocator = allocator<path_data::path_data_types>>
class path_factory;

class color_stop;
template <Allocator = allocator<color_stop>>
class color_stop_group;

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;
}
```

5 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 (3).

5.1 Enum class `io2d_error` [io2derror]

5.1.1 `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.

5.1.2 `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{ };
}
```

5.1.3 `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_factory` 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*]

5.2 Class `io2d_error_category` [io2derrcat]

5.2.1 `io2d_error_category` synopsis [io2derrcat.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class io2d_error_category : public std::error_category {
  public:
    // 5.2.3, 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;
  };

  // 5.2.4, 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;
}
```

5.2.2 `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.

5.2.3 `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*]

5.2.4 `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());`.

6 Colors [colors]

1 This section is forthcoming in a future revision.

6.1 Class `rgba_color` [rgbacolor]

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

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

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

4 Legal values for each channel are in the range $[0, 1]$.

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

6.1.1 `rgba_color` synopsis [rgbacolor.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class rgba_color {
    // 6.1.2, construct/copy/move/destroy:
        rgba_color(double r, double g, double b, double a = 1.0) noexcept;

    // 6.1.3, modifiers:
    void r(double val) noexcept;
    void g(double val) noexcept;
    void b(double val) noexcept;
    void a(double val) noexcept;

    // 6.1.4, observers:
    double r() const noexcept;
    double g() const noexcept;
    double b() const noexcept;
    double a() const noexcept;

    // 6.1.5, static member functions:
    static const rgba_color& alice_blue() noexcept;
    static const rgba_color& antique_white() noexcept;
    static const rgba_color& aqua() noexcept;
    static const rgba_color& aquamarine() noexcept;
    static const rgba_color& azure() noexcept;
    static const rgba_color& beige() noexcept;
    static const rgba_color& bisque() noexcept;
    static const rgba_color& black() noexcept;
    static const rgba_color& blanched_almond() noexcept;
    static const rgba_color& blue() noexcept;
    static const rgba_color& blue_violet() noexcept;
    static const rgba_color& brown() noexcept;
    static const rgba_color& burly_wood() noexcept;
    static const rgba_color& cadet_blue() noexcept;
    static const rgba_color& chartreuse() noexcept;
    static const rgba_color& chocolate() noexcept;
    static const rgba_color& coral() noexcept;
    static const rgba_color& cornflower_blue() noexcept;
```

```
static const rgba_color& cornsilk() noexcept;
static const rgba_color& crimson() noexcept;
static const rgba_color& cyan() noexcept;
static const rgba_color& dark_blue() noexcept;
static const rgba_color& dark_cyan() noexcept;
static const rgba_color& dark_goldenrod() noexcept;
static const rgba_color& dark_gray() noexcept;
static const rgba_color& dark_green() noexcept;
static const rgba_color& dark_grey() noexcept;
static const rgba_color& dark_khaki() noexcept;
static const rgba_color& dark_magenta() noexcept;
static const rgba_color& dark_olive_green() noexcept;
static const rgba_color& dark_orange() noexcept;
static const rgba_color& dark_orchid() noexcept;
static const rgba_color& dark_red() noexcept;
static const rgba_color& dark_salmon() noexcept;
static const rgba_color& dark_sea_green() noexcept;
static const rgba_color& dark_slate_blue() noexcept;
static const rgba_color& dark_slate_gray() noexcept;
static const rgba_color& dark_slate_grey() noexcept;
static const rgba_color& dark_turquoise() noexcept;
static const rgba_color& dark_violet() noexcept;
static const rgba_color& deep_pink() noexcept;
static const rgba_color& deep_sky_blue() noexcept;
static const rgba_color& dim_gray() noexcept;
static const rgba_color& dim_grey() noexcept;
static const rgba_color& dodger_blue() noexcept;
static const rgba_color& firebrick() noexcept;
static const rgba_color& floral_white() noexcept;
static const rgba_color& forest_green() noexcept;
static const rgba_color& fuchsia() noexcept;
static const rgba_color& gainsboro() noexcept;
static const rgba_color& ghost_white() noexcept;
static const rgba_color& gold() noexcept;
static const rgba_color& goldenrod() noexcept;
static const rgba_color& gray() noexcept;
static const rgba_color& green() noexcept;
static const rgba_color& green_yellow() noexcept;
static const rgba_color& grey() noexcept;
static const rgba_color& honeydew() noexcept;
static const rgba_color& hot_pink() noexcept;
static const rgba_color& indian_red() noexcept;
static const rgba_color& indigo() noexcept;
static const rgba_color& ivory() noexcept;
static const rgba_color& khaki() noexcept;
static const rgba_color& lavender() noexcept;
static const rgba_color& lavender_blush() noexcept;
static const rgba_color& lawn_green() noexcept;
static const rgba_color& lemon_chiffon() noexcept;
static const rgba_color& light_blue() noexcept;
static const rgba_color& light_coral() noexcept;
static const rgba_color& light_cyan() noexcept;
static const rgba_color& light_goldenrod_yellow() noexcept;
static const rgba_color& light_gray() noexcept;
static const rgba_color& light_green() noexcept;
```

```
static const rgba_color& light_grey() noexcept;
static const rgba_color& light_pink() noexcept;
static const rgba_color& light_salmon() noexcept;
static const rgba_color& light_sea_green() noexcept;
static const rgba_color& light_sky_blue() noexcept;
static const rgba_color& light_slate_gray() noexcept;
static const rgba_color& light_slate_grey() noexcept;
static const rgba_color& light_steel_blue() noexcept;
static const rgba_color& light_yellow() noexcept;
static const rgba_color& lime() noexcept;
static const rgba_color& lime_green() noexcept;
static const rgba_color& linen() noexcept;
static const rgba_color& magenta() noexcept;
static const rgba_color& maroon() noexcept;
static const rgba_color& medium_aquamarine() noexcept;
static const rgba_color& medium_blue() noexcept;
static const rgba_color& medium_orchid() noexcept;
static const rgba_color& medium_purple() noexcept;
static const rgba_color& medium_sea_green() noexcept;
static const rgba_color& medium_slate_blue() noexcept;
static const rgba_color& medium_spring_green() noexcept;
static const rgba_color& medium_turquoise() noexcept;
static const rgba_color& medium_violet_red() noexcept;
static const rgba_color& midnight_blue() noexcept;
static const rgba_color& mint_cream() noexcept;
static const rgba_color& misty_rose() noexcept;
static const rgba_color& moccasin() noexcept;
static const rgba_color& navajo_white() noexcept;
static const rgba_color& navy() noexcept;
static const rgba_color& old_lace() noexcept;
static const rgba_color& olive() noexcept;
static const rgba_color& olive_drab() noexcept;
static const rgba_color& orange() noexcept;
static const rgba_color& orange_red() noexcept;
static const rgba_color& orchid() noexcept;
static const rgba_color& pale_goldenrod() noexcept;
static const rgba_color& pale_green() noexcept;
static const rgba_color& pale_turquoise() noexcept;
static const rgba_color& pale_violet_red() noexcept;
static const rgba_color& papaya_whip() noexcept;
static const rgba_color& peach_puff() noexcept;
static const rgba_color& peru() noexcept;
static const rgba_color& pink() noexcept;
static const rgba_color& plum() noexcept;
static const rgba_color& powder_blue() noexcept;
static const rgba_color& purple() noexcept;
static const rgba_color& red() noexcept;
static const rgba_color& rosy_brown() noexcept;
static const rgba_color& royal_blue() noexcept;
static const rgba_color& saddle_brown() noexcept;
static const rgba_color& salmon() noexcept;
static const rgba_color& sandy_brown() noexcept;
static const rgba_color& sea_green() noexcept;
static const rgba_color& sea_shell() noexcept;
static const rgba_color& sienna() noexcept;
```

```
    static const rgba_color& silver() noexcept;
    static const rgba_color& sky_blue() noexcept;
    static const rgba_color& slate_blue() noexcept;
    static const rgba_color& slate_gray() noexcept;
    static const rgba_color& slate_grey() noexcept;
    static const rgba_color& snow() noexcept;
    static const rgba_color& spring_green() noexcept;
    static const rgba_color& steel_blue() noexcept;
    static const rgba_color& tan() noexcept;
    static const rgba_color& teal() noexcept;
    static const rgba_color& thistle() noexcept;
    static const rgba_color& tomato() noexcept;
    static const rgba_color& transparent_black() noexcept;
    static const rgba_color& turquoise() noexcept;
    static const rgba_color& violet() noexcept;
    static const rgba_color& wheat() noexcept;
    static const rgba_color& white() noexcept;
    static const rgba_color& white_smoke() noexcept;
    static const rgba_color& yellow() noexcept;
    static const rgba_color& yellow_green() noexcept;
  };

  // 6.1.6, non-member operators:
  bool operator==(const rgba_color& lhs, const rgba_color& rhs) noexcept;
  bool operator!=(const rgba_color& lhs, const rgba_color& rhs) noexcept;
} } } }
```

6.1.2 `rgba_color` constructors and assignment operators [rgbacolor.cons]

```
rgba_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 `rgba_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`.

6.1.3 `rgba_color` modifiers [rgbacolor.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`.

6.1.4 `rgba_color` observers [rgbacolor.observers]

```
double r() const noexcept;
```

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

```
double g() const noexcept;
```

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

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

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

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

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

6.1.5 `rgba_color` static member functions [rgbacolor.statics]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
static const rgba_color& floral_white() noexcept;
```

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

```
static const rgba_color& forest_green() noexcept;
```

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

```
static const rgba_color& fuchsia() noexcept;
```

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

```
static const rgba_color& gainsboro() noexcept;
```

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

```
static const rgba_color& ghost_white() noexcept;
```

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

```
static const rgba_color& gold() noexcept;
```

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

```
static const rgba_color& goldenrod() noexcept;
```

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

```
static const rgba_color& gray() noexcept;
```

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

```
static const rgba_color& green() noexcept;
```

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

```
static const rgba_color& green_yellow() noexcept;
```

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

```
static const rgba_color& grey() noexcept;
```

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

```
static const rgba_color& honeydew() noexcept;
```

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

```
static const rgba_color& hot_pink() noexcept;
```

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

```
static const rgba_color& indian_red() noexcept;
```

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

```
static const rgba_color& indigo() noexcept;
```

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

```
static const rgba_color& ivory() noexcept;
```

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

```
static const rgba_color& khaki() noexcept;
```

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

```
static const rgba_color& lavender() noexcept;
```

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

```
static const rgba_color& lavender_blush() noexcept;
```

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

```
static const rgba_color& lawn_green() noexcept;
```

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

```
static const rgba_color& lemon_chiffon() noexcept;
```

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

```
static const rgba_color& light_blue() noexcept;
```

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

```
static const rgba_color& light_coral() noexcept;
```

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

```
static const rgba_color& light_cyan() noexcept;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
static const rgba_color& lime() noexcept;
```

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

```
static const rgba_color& lime_green() noexcept;
```

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

```
static const rgba_color& linen() noexcept;
```

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

```
static const rgba_color& magenta() noexcept;
```

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

```
static const rgba_color& maroon() noexcept;
```

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

```
static const rgba_color& medium_aquamarine() noexcept;
```

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

```
static const rgba_color& medium_blue() noexcept;
```

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

```
static const rgba_color& medium_orchid() noexcept;
```

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

```
static const rgba_color& medium_purple() noexcept;
```

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

```
static const rgba_color& medium_sea_green() noexcept;
```

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

```
static const rgba_color& medium_slate_blue() noexcept;
```

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

```
static const rgba_color& medium_spring_green() noexcept;
```

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

```
static const rgba_color& medium_turquoise() noexcept;
```

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

```
static const rgba_color& medium_violet_red() noexcept;
```

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

```
static const rgba_color& midnight_blue() noexcept;
```

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

```
static const rgba_color& mint_cream() noexcept;
```

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

```
static const rgba_color& misty_rose() noexcept;
```

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

```
static const rgba_color& moccasin() noexcept;
```

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

```
static const rgba_color& navajo_white() noexcept;
```

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

```
static const rgba_color& navy() noexcept;
```

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

```
static const rgba_color& old_lace() noexcept;
```

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

```
static const rgba_color& olive() noexcept;
```

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

```
static const rgba_color& olive_drab() noexcept;
```

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

```
static const rgba_color& orange() noexcept;
```

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

```
static const rgba_color& orange_red() noexcept;
```

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

```
static const rgba_color& orchid() noexcept;
```

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

```
static const rgba_color& pale_goldenrod() noexcept;
```

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

```
static const rgba_color& pale_green() noexcept;
```

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

```
static const rgba_color& pale_turquoise() noexcept;
```

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

```
static const rgba_color& pale_violet_red() noexcept;
```

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

```
static const rgba_color& papaya_whip() noexcept;
```

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

```
static const rgba_color& peach_puff() noexcept;
```

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

```
static const rgba_color& peru() noexcept;
```

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

```
static const rgba_color& pink() noexcept;
```

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

```
static const rgba_color& plum() noexcept;
```

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

```
static const rgba_color& powder_blue() noexcept;
```

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

```
static const rgba_color& purple() noexcept;
```

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

```
static const rgba_color& red() noexcept;
```

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

```
static const rgba_color& rosy_brown() noexcept;
```

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

```
static const rgba_color& royal_blue() noexcept;
```

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

```
static const rgba_color& saddle_brown() noexcept;
```

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

```
static const rgba_color& salmon() noexcept;
```

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

```
static const rgba_color& sandy_brown() noexcept;
```

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

```
static const rgba_color& sea_green() noexcept;
```

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

```
static const rgba_color& sea_shell() noexcept;
```

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

```
static const rgba_color& sienna() noexcept;
```

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

```
static const rgba_color& silver() noexcept;
```

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

```
static const rgba_color& sky_blue() noexcept;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
static const rgba_color& violet() noexcept;
```

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

```
static const rgba_color& wheat() noexcept;
```

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

```
static const rgba_color& white() noexcept;
```

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

```
static const rgba_color& white_smoke() noexcept;
```

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

```
static const rgba_color& yellow() noexcept;
```

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

```
static const rgba_color& yellow_green() noexcept;
```

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

6.1.6 `rgba_color` non-member operators [rgbacolor.ops]

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

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

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

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

6.2 `literals` namespace [literals]

6.2.1 `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);
  }
} } } }
```

6.2.2 `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`

7 Geometry [geometry]

1 This section contains classes that define or manipulate geometry.

7.1 Class `vector_2d` [vector2d]

7.1.1 `vector_2d` Description [vector2d.intro]

1 The class `vector_2d` describes an object that stores a two-dimensional Euclidean vector.

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

7.1.2 `vector_2d` synopsis [vector2d.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class vector_2d {
  public:
    // 7.1.3, construct/copy/move/destroy:
    vector_2d(double x, double y) noexcept;

    // 7.1.4, modifiers:
    void x(double value) noexcept;
    void y(double value) noexcept;

    // 7.1.5, observers:
    double x() const noexcept;
    double y() const noexcept;
    double length() const noexcept;
    double dot(const vector_2d& other) const noexcept;
    vector_2d to_unit() const noexcept;

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

  // 7.1.7, non-member operators:
  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;
} } } }
```

7.1.3 `vector_2d` constructors and assignment operators [vector2d.cons]

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

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

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

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

7.1.4 `vector_2d` modifiers [vector2d.modifiers]

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

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

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

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

7.1.5 `vector_2d` observers [vector2d.observers]

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

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

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

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

```
double length() const noexcept;
```

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

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

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

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

5 *Returns:* `vector_2d{ *this.x() / length(), *this.y() / length()}`.

7.1.6 `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`.

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

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

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

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

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

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

3 *Returns:* `vector_2d(lhs)`.

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

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

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

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

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

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

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

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

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

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

7.2 Class `rectangle` [rectangle]

7.2.1 `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`.

7.2.2 `rectangle` synopsis [rectangle.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class rectangle {
  public:
    // 7.2.3, construct/copy/move/destroy:
    rectangle(double x, double y, double width, double height) noexcept;
    rectangle(const vector_2d& tl, const vector_2d& br) noexcept;

    // 7.2.4, modifiers:
    void x(double val) noexcept;
    void y(double val) noexcept;
    void width(double val) noexcept;
    void height(double val) noexcept;
    void top_left(const vector_2d& val) noexcept;
    void bottom_right(const vector_2d& val) noexcept;

    // 7.2.5, observers:
    double x() const noexcept;
    double y() const noexcept;
    double width() const noexcept;
    double height() const noexcept;
    vector_2d top_left() const noexcept;
    vector_2d bottom_right() const noexcept;
  };
} } } }
```

7.2.3 `rectangle` constructors [rectangle.cons]

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

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

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

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

4 The width shall be set to the value of `w`.

5 The height shall be set to the value of `h`.

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

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

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

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

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

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

7.2.4 `rectangle` modifiers [rectangle.modifiers]

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

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

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

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

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

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

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

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

```
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()`.

```
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())`.

7.2.5 `rectangle` observers [rectangle.observers]

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

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

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

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

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

3 *Returns:* The value of the width.

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

4 *Returns:* The value of the height.

```
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.

```
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.

7.3 Class `circle` [circle]

7.3.1 `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`.

7.3.2 `circle` synopsis [circle.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class circle {
  public:
    // 7.3.3, construct/copy/move/destroy:
    rectangle(const vector_2d& ctr, double rad) noexcept;

    // 7.3.4, modifiers:
    void center(const vector_2d& ctr) noexcept;
    void radius(double rad) noexcept;

    // 7.3.5, observers:
    vector_2d center() const noexcept;
    double radius() const noexcept;
  };
} } } }
```

7.3.3 `circle` constructors and assignment operators [circle.cons]

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

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

2 The center shall have the value of `ctr`.

3 The radius shall have the value of `rad`.

7.3.4 `circle` modifiers [circle.modifiers]

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

1 *Effects:* The center shall have the value of `ctr`.

```
void radius(double rad) noexcept;
```

2 *Effects:* The radius shall have the value of `rad`.

7.3.5 `circle` observers [circle.observers]

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

1 *Returns:* The value of the center.

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

2 *Returns:* THe value of the radius.

7.4 Class `matrix_2d` [matrix2d]

7.4.1 `matrix_2d` Description [matrix2d.intro]

1 The `matrix_2d` class represents a two-dimensional, three row by three column, row-major 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 [*Note:* If the third column's data members were included, 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 resulting matrix would be as such:
[[$m00$ $m01$ $m02$]]
[[$m10$ $m11$ $m12$]]
[[$m20$ $m21$ $m22$]] — *end note*]

7.4.2 `matrix_2d` synopsis [matrix2d.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class matrix_2d {
  public:
    // 7.4.3, construct/copy/move/destroy:
    matrix_2d() noexcept;
    matrix_2d(const matrix_2d&) noexcept;
    matrix_2d& operator=(const matrix_2d&) noexcept;
    matrix_2d(matrix_2d&&) noexcept;
    matrix_2d& operator=(matrix_2d&&) noexcept;
    matrix_2d(double v00, double v01, double v10, double v11,
      double v20, double v21) noexcept;

    // 7.4.4, static factory functions:
    static matrix_2d init_identity() noexcept;
    static matrix_2d init_translate(const vector_2d& value) noexcept;
    static matrix_2d init_scale(const vector_2d& value) noexcept;
    static matrix_2d init_rotate(double radians) noexcept;
    static matrix_2d init_shear_x(double factor) noexcept;
    static matrix_2d init_shear_y(double factor) noexcept;

    // 7.4.5, modifiers:
    void m00(double value) noexcept;
```

```
    void m01(double value) noexcept;
    void m10(double value) noexcept;
    void m11(double value) noexcept;
    void m20(double value) noexcept;
    void m21(double value) noexcept;
    matrix_2d& translate(const vector_2d& value) noexcept;
    matrix_2d& scale(const vector_2d& value) noexcept;
    matrix_2d& rotate(double radians) noexcept;
    matrix_2d& shear_x(double factor) noexcept;
    matrix_2d& shear_y(double factor) noexcept;
    matrix_2d& invert();
    matrix_2d& invert(error_code& ec) noexcept;

    // 7.4.6, observers:
    double m00() const noexcept;
    double m01() const noexcept;
    double m10() const noexcept;
    double m11() const noexcept;
    double m20() const noexcept;
    double m21() const noexcept;
    bool is_invertible const noexcept;
    double determinant() const;
    double determinant(error_code& ec) const noexcept;
    vector_2d transform_distance(const vector_2d& dist) const noexcept;
    vector_2d transform_point(const vector_2d& pt) const noexcept;

    // 7.4.7, matrix_2d member operators:
    matrix_2d& operator*=(const matrix_2d& rhs) noexcept;
  };

  // 7.4.8, matrix_2d non-member operators:
  matrix_2d operator*(const matrix_2d& lhs, const matrix_2d& rhs) noexcept;
  bool operator==(const matrix_2d& lhs, const matrix_2d& rhs) noexcept;
  bool operator!=(const matrix_2d& lhs, const matrix_2d& rhs) noexcept;
} } } }
```

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

```
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()`.

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

Effects: Constructs an object of type `matrix_2d`.
The m00 value shall be set to the value of `v00`.
The m01 value shall be set to the value of `v01`.
The m10 value shall be set to the value of `v10`.
The m11 value shall be set to the value of `v11`.
The m20 value shall be set to the value of `v20`.
The m21 value shall be set to the value of `v21`.

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

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

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

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

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

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

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

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

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

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

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

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

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

7.4.5 `matrix_2d` modifiers [matrix2d.modifiers]

```
void m00(double val) noexcept;
```

Effects: The m00 value shall be set to the value of `val`.

```
void m01(double val) noexcept;
```

Effects: The m01 value shall be set to the value of `val`.

```
void m10(double val) noexcept;
```

Effects: The m10 value shall be set to the value of `val`.

```
void m11(double val) noexcept;
```

Effects: The m11 value shall be set to the value of `val`.

```
void m20(double val) noexcept;
```

Effects: The m20 value shall be set to the value of `val`.

```
void m21(double value) noexcept;
```

Effects: The m21 value shall be set to the value of `val`.

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

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

Returns: `*this`.

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

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

Returns: `*this`.

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

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

Returns: `*this`.

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

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

Returns: `*this`.

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

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

Returns: `*this`.

```
matrix_2d& invert();
matrix_2d& invert(error_code& ec) noexcept;
```

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);
```

Returns: `*this`.

Throws: As specified in Error reporting (3).

Error conditions: `io2d_error::invalid_matrix` if `this->is_invertible() == false`.

Remark: If an error occurs, this function shall have no effects.

7.4.6 `matrix_2d` observers [matrix2d.observers]

```
double m00() const noexcept;
```

1 *Returns:* The m00 value.

```
double m01() const noexcept;
```

2 *Returns:* The m01 value.

```
double m10() const noexcept;
```

3 *Returns:* The m10 value.

```
double m11() const noexcept;
```

4 *Returns:* The m11 value.

```
double m20() const noexcept;
```

5 *Returns:* The m20 value.

```
double m21() const noexcept;
```

6 *Returns:* The m21 value.

```
bool is_invertible 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())`
(7.7) — `(*this.m00() * *this.m11() - *this.m01() * *this.m10()) != 0.0`

8 Otherwise returns `false`.

```
double determinant() const;
double determinant(error_code& ec) const noexcept;
```

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

10 In the event of a non-throwing error, the function shall return `numeric_limits<double>::quiet_NaN()`.

11 *Throws:* As specified in Error reporting (3).

12 *Error conditions:* `io2d_error::invalid_matrix` if `!isfinite(*this.m00()) || !isfinite(*this.m01()) || !isfinite(*this.m10()) || !isfinite(*this.m11()) || !isfinite(*this.m20()) || !isfinite(*this.m2`

```
vector_2d transform_distance(const vector_2d& dist) const noexcept;
```

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

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

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

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

7.4.7 `matrix_2d` member operators [matrix2d.member.ops]

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

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

2 *Returns:* `*this`

7.4.8 `matrix_2d` non-member operators [matrix2d.ops]

```
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()
}
```

```
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()`.

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

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

8 Paths [paths]

1 Paths define geometric objects which can be stroked (Table 17), filled, masked, and used to define or modify a Clip Area (Table 16).

2 Paths are created using a `path_factory` 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 (10.13.1) 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, which are set forth below.

5 A `path_group` object is an immutable resource wrapper containing a path group (8.16). A `path_group` object is created from a `path_factory` object. It can also be default constructed, in which case the `path_group` object contains no paths.

8.1 Processing paths [paths.processing]

1 This section is normative. It describes how to convert the path group of a properly formed `path_factory` object from a collection of `path_data::data` objects to a collection of `path_data::data` objects which have had their points transformed in accordance with the origin and transformation matrix of the `path_factory` object and any `path_data::change_origin` and `path_data::change_matrix` objects in the path group of the `path_factory` object.

2 The following code shows how to properly process a `path_factory` object `p` and store the results into a `vector<path_data::data>`:

```
#include <cmath>
#include <vector>
#include <variant>
#include <experimental/io2d>
using namespace std;
using namespace std::experimental::io2d;

matrix_2d m;
vector_2d origin;
vector_2d currentPoint; // Tracks the untransformed current point.
bool hasCurrentPoint = false;
vector_2d closePoint;   // Tracks the transformed close point.
vector<path_data::path_data_types> v;

for (auto val : p) {
  std::visit([&](auto&& item) {
    using T = std::remove_cv_t<std::remove_reference_t<decltype item>>;

    if constexpr(std::is_same_v<T, path_data::abs_move>) {
      currentPoint = item.to();
      auto pt = m.transform_point(currentPoint - origin) + origin;
      hasCurrentPoint = true;
      v.emplace_back(in_place_type_t<path_data::abs_move>, pt);
      closePoint = pt
```

```
}
else if constexpr(std::is_same_v<T, path_data::abs_line>) {
  currentPoint = item.to();
  auto pt = m.transform_point(currentPoint - origin) + origin;
  if (hasCurrentPoint) {
    v.emplace_back(in_place_type_t<path_data::abs_line>, pt);
  }
  else {
    v.emplace_back(in_place_type_t<path_data::abs_move>, pt);
    v.emplace_back(in_place_type_t<path_data::abs_line>, pt);
    hasCurrentPoint = true;
    closePoint = pt;
  }
}
else if constexpr(std::is_same_v<T, path_data::abs_cubic_curve>) {
  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 (!hasCurrentPoint) {
    currentPoint = item.control_point_1();
    v.emplace_back(in_place_type_t<path_data::abs_move>, pt1);
    hasCurrentPoint = true;
    closePoint = pt1;
  }
  v.emplace_back(in_place_type_t<path_data::abs_cubic_curve>, pt1,
    pt2, pt3);
  currentPoint = item.end_point();
}
else if constexpr(std::is_same_v<T,
  path_data::abs_quadratic_curve>) {
  auto pt1 = m.transform_point(item.control_point() - origin) + origin;
  auto pt2 = m.transform_point(item.end_point() - origin) + origin;
  if (!hasCurrentPoint) {
          currentPoint = item.control_point();
          v.emplace_back(in_place_type_t<path_data::abs_move>, pt1);
          hasCurrentPoint = true;
          closePoint = pt1;
  }
  v.emplace_back(in_place_type_t<path_data::abs_quadratic_curve>,
    pt1, pt2);
  currentPoint = item.end_point();
}
else if constexpr(std::is_same_v<T, path_data::new_path>) {
  hasCurrentPoint = false;
  v.emplace_back(in_place_type_t<path_data::new_path>);
}
else if constexpr(std::is_same_v<T, path_data::close_path>) {
  if (hasCurrentPoint) {
    v.emplace_back(in_place_type_t<path_data::close_path>);
    v.emplace_back(in_place_type_t<path_data::abs_move>,
      closePoint);
    auto invM = matrix_2d{m}.invert();
    // Need to assign the untransformed closePoint value to currentPoint.
    currentPoint = invM.transform_point(closePoint - origin) + origin;
  }
```

```
}
else if constexpr(std::is_same_v<T, path_data::rel_move>) {
  currentPoint = item.to() + currentPoint;
  auto pt = m.transform_point(currentPoint - origin) + origin;
  v.emplace_back(in_place_type_t<path_data::abs_move>, pt);
  hasCurrentPoint = true;
  closePoint = pt
  n.close_point(pt);
}
else if constexpr(std::is_same_v<T, path_data::rel_line>) {
  currentPoint = item.to() + currentPoint;
  auto pt = m.transform_point(currentPoint - origin) + origin;
  v.emplace_back(in_place_type_t<path_data::abs_line>, pt);
}
else if constexpr(std::is_same_v<T, path_data::rel_cubic_curve>) {
  auto pt1 = m.transform_point(item.control_point_1() + currentPoint -
  origin) + origin;
  auto pt2 = m.transform_point(item.control_point_2() + currentPoint -
  origin) + origin;
  auto pt3 = m.transform_point(item.end_point() + currentPoint - origin) +
    origin;
  v.emplace_back(in_place_type_t<path_data::abs_cubic_curve>,
    pt1, pt2, pt3);
  currentPoint = item.end_point() + currentPoint;
}
else if constexpr(std::is_same_v<T,
path_data::rel_quadratic_curve>) {
  auto pt1 = m.transform_point(item.control_point() + currentPoint -
    origin) + origin;
  auto pt2 = m.transform_point(item.end_point() + currentPoint -
    origin) + origin;
  v.emplace_back(in_place_type_t<path_data::rel_quadratic_curve>,
  pt1, pt2);
  currentPoint = item.end_point() + currentPoint;
}
else if constexpr(std::is_same_v<T, path_data::arc_clockwise>) {
  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 >= halfpi) {
    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 (hasCurrentPoint) {
      currentPoint = startPt;
      auto pt = m.transform_point(currentPoint - origin) + origin;
      v.emplace_back(in_place_type_t<path_data::abs_line>, pt);
    }
    else {
      currentPoint = startPt;
      auto pt = m.transform_point(currentPoint - origin) + origin;
      v.emplace_back(in_place_type_t<path_data::abs_move>, pt);
      hasCurrentPoint = true;
      closePt = 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_t<path_data::abs_cubic_curve>, cpt1,
        cpt2, cpt3);
      currTheta += theta;
    }
  }
  else if constexpr(std::is_same_v<T, path_data::arc_counterclockwise>) {
  {
    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 >= halfpi) {
  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 (hasCurrentPoint) {
  currentPoint = startPt;
  auto pt = m.transform_point(currentPoint - origin) + origin;
  v.emplace_back(in_place_type_t<path_data::abs_line>, pt);
}
else {
  currentPoint = startPt;
  auto pt = m.transform_point(currentPoint - origin) + origin;
  v.emplace_back(in_place_type_t<path_data::abs_move>, pt);
  hasCurrentPoint = true;
  closePt = 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_t<path_data::cubic_curve_to>, cpt1,
            cpt2, cpt3);
          currTheta -= theta;
        }
      }
      else if constexpr(std::is_same_v<T, path_data::change_matrix>) {
        m = item.matrix();
      }
      else if constexpr(std::is_same_v<T, path_data::change_origin>) {
        origin = item.origin();
      }
    }, val);
  }
```

8.2 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.

8.2.1 `new_path` synopsis [newpath.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class new_path {
    };
  };
} } } }
```

8.3 Class `close_path` [closepath]

1 This class is a path instruction that contains no data. It creates a closed path within a path group.

8.3.1 `close_path` synopsis [closepath.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class close_path {
    };
};
} } } }
```

8.4 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.

8.4.1 `abs_move` synopsis [absmove.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_move {
    public:
      // 8.4.2, construct:
      explicit abs_move(const vector_2d& pt) noexcept;

      // 8.4.3, modifiers:
      void to(const vector_2d& pt) noexcept;

      // 8.4.4, observers:
      vector_2d to() const noexcept;
    };
  };
} } } }
```

8.4.2 `abs_move` constructors [absmove.cons]

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

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

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

8.4.3 `abs_move` modifiers [absmove.modifiers]

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

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

8.4.4 `abs_move` observers [absmove.observers]

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

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

8.5 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.

8.5.1 `rel_move` synopsis [relmove.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_move {
    public:
      // 8.5.2, construct:
      explicit rel_move(const vector_2d& pt) noexcept;

      // 8.5.3, modifiers:
      void to(const vector_2d& pt) noexcept;
```

```
      // 8.5.4, observers:
      vector_2d to() const noexcept;
    };
  };
} } } }
```

8.5.2 `rel_move` constructors [relmove.cons]

```
explicit rel_move(const vector_2d& pt) noexcept;
```

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

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

8.5.3 `rel_move` modifiers [relmove.modifiers]

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

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

8.5.4 `rel_move` observers [relmove.observers]

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

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

8.6 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`.

8.6.1 `abs_line` synopsis [absline.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_line {
    public:
      // 8.6.2, construct:
      explicit abs_line(const vector_2d& pt) noexcept;

      // 8.6.3, modifiers:
      void to(const vector_2d& pt) noexcept;

      // 8.6.4, observers:
      vector_2d to() const noexcept;
    };
  };
} } } }
```

8.6.2 `abs_line` constructors and assignment operators [absline.cons]

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

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

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

8.6.3 `abs_line` modifiers [absline.modifiers]

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

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

8.6.4 `abs_line` observers [absline.observers]

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

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

8.7 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.

8.7.1 `rel_line` synopsis [relline.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_line {
    public:
      // 8.7.2, construct:
      explicit rel_line(const vector_2d& pt) noexcept;

      // 8.7.3, modifiers:
      void to(const vector_2d& pt) noexcept;

      // 8.7.4, observers:
      vector_2d to() const noexcept;
    };
  };
} } } }
```

8.7.2 `rel_line` constructors [relline.cons]

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

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

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

8.7.3 `rel_line` modifiers [relline.modifiers]

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

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

8.7.4 `rel_line` observers [relline.observers]

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

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

8.8 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`.

8.8.1 `abs_cubic_curve` synopsis [abscubiccurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_cubic_curve {
    public:
      // 8.8.2, construct:
      abs_cubic_curve(const vector_2d& cp1, const vector_2d& cp2,
        const vector_2d& ep) noexcept;

      // 8.8.3, modifiers:
      void control_point_1(const vector_2d& cp) noexcept;
      void control_point_2(const vector_2d& cp) noexcept;
      void end_point(const vector_2d& ep) noexcept;


      // 8.8.4, observers:
      vector_2d control_point_1() const noexcept;
      vector_2d control_point_2() const noexcept;
      vector_2d end_point() const noexcept;
    };
  };
} } } }
```

8.8.2 `abs_cubic_curve` constructors [abscubiccurve.cons]

```
abs_cubic_curve(const vector_2d& cp1, const vector_2d& cp2,
  const vector_2d& ep) noexcept;
```

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

2 The first control point shall be set to the value of `cp1`.

3 The second control point shall be set to the value of `cp2`.

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

8.8.3 `abs_cubic_curve` modifiers [abscubiccurve.modifiers]

```
void control_point_1(const vector_2d& cp) noexcept;
```

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

```
void control_point_2(const vector_2d& cp) noexcept;
```

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

```
void end_point(const vector_2d& ep) noexcept;
```

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

8.8.4 `abs_cubic_curve` observers [abscubiccurve.observers]

```
vector_2d control_point_1() const noexcept;
```

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

```
vector_2d control_point_2() const noexcept;
```

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

```
vector_2d end_point() const noexcept;
```

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

8.9 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.

8.9.1 `rel_cubic_curve` synopsis [relcubiccurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_cubic_curve {
    public:
      // 8.9.2, construct
      rel_cubic_curve(const vector_2d& cp1, const vector_2d& cp2,
        const vector_2d& ep) noexcept;

      // 8.9.3, modifiers:
      void control_point_1(const vector_2d& cp) noexcept;
      void control_point_2(const vector_2d& cp) noexcept;
      void end_point(const vector_2d& ep) noexcept;

      // 8.9.4, observers:
      vector_2d control_point_1() const noexcept;
      vector_2d control_point_2() const noexcept;
      vector_2d end_point() const noexcept;
    };
  };
} } } }
```

8.9.2 `rel_cubic_curve` constructors [relcubiccurve.cons]

```
rel_cubic_curve(const vector_2d& cp1, const vector_2d& cp2,
  const vector_2d& ep) noexcept;
```

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

2 The first control point shall be set to the value of `cp1`.

3 The second control point shall be set to the value of `cp2`.

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

8.9.3 `rel_cubic_curve` modifiers [relcubiccurve.modifiers]

```
void control_point_1(const vector_2d& cp) noexcept;
```

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

```
void control_point_2(const vector_2d& value) noexcept;
```

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

```
void end_point(const vector_2d& value) noexcept;
```

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

8.9.4 `rel_cubic_curve` observers [relcubiccurve.observers]

```
vector_2d control_point_1() const noexcept;
```

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

```
vector_2d control_point_2() const noexcept;
```

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

```
vector_2d end_point() const noexcept;
```

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

8.10 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`.

8.10.1 `abs_quadratic_curve` synopsis [absquadraticcurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class abs_cubic_curve {
    public:
      // 8.10.2, construct:
      abs_quadratic_curve(const vector_2d& cp, const vector_2d& ep) noexcept;

      // 8.10.3, modifiers:
      void control_point(const vector_2d& cp) noexcept;
      void end_point(const vector_2d& ep) noexcept;

      // 8.10.4, observers:
      vector_2d control_point() const noexcept;
      vector_2d end_point() const noexcept;
    };
  };
} } } }
```

8.10.2 `abs_quadratic_curve` constructors [absquadraticcurve.cons]

```
abs_quadratic_curve(const vector_2d& cp, const vector_2d& ep) noexcept;
```

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

2 The control point shall be set to the value of `cp`.

3 The end point shall be set to the value of `ep`.

8.10.3 `abs_cubic_curve` modifiers [absquadraticcurve.modifiers]

```
void control_point(const vector_2d& cp) noexcept;
```

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

```
void end_point(const vector_2d& ep) noexcept;
```

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

8.10.4 `abs_quadratic_curve` observers [absquadraticcurve.observers]

```
vector_2d control_point() const noexcept;
```

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

```
vector_2d end_point() const noexcept;
```

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

8.11 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.

8.11.1 `rel_quadratic_curve` synopsis [relquadraticcurve.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class rel_cubic_curve {
    public:
      // 8.11.2, construct:
      rel_quadratic_curve(const vector_2d& cp, const vector_2d& ep) noexcept;

      // 8.11.3, modifiers:
      void control_point(const vector_2d& cp) noexcept;
      void end_point(const vector_2d& ep) noexcept;

      // 8.11.4, observers:
      vector_2d control_point() const noexcept;
      vector_2d end_point() const noexcept;
    };
  };
} } } }
```

8.11.2 `rel_quadratic_curve` constructors [relquadraticcurve.cons]

```
rel_quadratic_curve(const vector_2d& cp, const vector_2d& ep) noexcept;
```

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

2 The control point shall be set to the value of `cp`.

3 The end point shall be set to the value of `ep`.

8.11.3 `rel_cubic_curve` modifiers [relquadraticcurve.modifiers]

```
void control_point(const vector_2d& cp) noexcept;
```

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

```
void end_point(const vector_2d& ep) noexcept;
```

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

8.11.4 `rel_quadratic_curve` observers [relquadraticcurve.observers]

```
vector_2d control_point() const noexcept;
```

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

```
vector_2d end_point() const noexcept;
```

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

8.12 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 center of type `vector_2d`, a radius of type `double`, 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 8.1, `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*]

8.12.1 `arc_clockwise` synopsis [arcclockwise.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class arc_clockwise {
    public:
      // 8.12.2, construct/copy/move/destroy:
      arc_clockwise(const vector_2d& ctr, double rad, double angle1,
        double angle2) noexcept;

      // 8.12.3, modifiers:
      void center(const vector_2d& ctr) noexcept;
      void radius(double r) noexcept;
      void angle_1(double radians) noexcept;
      void angle_2(double radians) noexcept;

      // 8.12.4, observers:
      vector_2d center() const noexcept;
      double radius() const noexcept;
      double angle_1() const noexcept;
      double angle_2() const noexcept;
    };
  };
} } } }
```

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

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

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

2 The center shall be set to the value of `ctr`.

3 The radius shall be set to the value of `rad`.

4 The first angle shall be set to the value of `angle1`.

5 The second angle shall be set to the value of `angle2`.

8.12.3 `arc_clockwise` modifiers [arcclockwise.modifiers]

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

1 *Effects:* The center shall be set to the value of `ctr`.

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

2 *Effects:* The radius shall be set to the value of `r`.

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

3 *Effects:* The first angle shall be set to the value of `radians`.

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

4 *Effects:* The second angle shall be set to the value of `radians`.

8.12.4 `arc_clockwise` observers [arcclockwise.observers]

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

1 *Returns:* The value of the center.

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

2 *Returns:* The value of the radius.

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

3 *Returns:* The value of the first angle.

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

4 *Returns:* The value of the second angle.

8.13 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 center of type `vector_2d`, a radius of type `double`, 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 8.1, `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*]

8.13.1 `arc_counterclockwise` synopsis [arccounterclockwise.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class arc_counterclockwise {
    public:
      // 8.13.2, construct:
      arc_counterclockwise(const vector_2d& ctr, double rad, double angle1,
        double angle2) noexcept;

      // 8.13.3, modifiers:
      void center(const vector_2d& ctr) noexcept;
      void radius(double r) noexcept;
      void angle_1(double radians) noexcept;
      void angle_2(double radians) noexcept;

      // 8.13.4, observers:
```

```
      vector_2d center() const noexcept;
      double radius() const noexcept;
      double angle_1() const noexcept;
      double angle_2() const noexcept;
    };
  };
} } } }
```

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

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

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

2 The center shall be set to the value of `ctr`.

3 The radius shall be set to the value of `rad`.

4 The first angle shall be set to the value of `angle1`.

5 The second angle shall be set to the value of `angle2`.

8.13.3 `arc_counterclockwise` modifiers [arccounterclockwise.modifiers]

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

1 *Effects:* The center shall be set to the value of `ctr`.

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

2 *Effects:* The radius shall be set to the value of `r`.

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

3 *Effects:* The first angle shall be set to the value of `radians`.

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

4 *Effects:* The second angle shall be set to the value of `radians`.

8.13.4 `arc_counterclockwise` observers [arccounterclockwise.observers]

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

1 *Returns:* The value of the center.

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

2 *Returns:* The value of the radius.

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

3 *Returns:* The value of the first angle.

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

4 *Returns:* The value of the second angle.

8.14 Class `change_matrix` [changematrix]

8.14.1 `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 8.1.

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:
      // 8.14.2, construct:
      explicit change_matrix(const matrix_2d& m) noexcept;

      // 8.14.3, modifiers:
      void matrix(const matrix_2d& m) noexcept;

      // 8.14.4, observers:
      matrix_2d matrix() const noexcept;
    };
  };
} } } }
```

8.14.2 `change_matrix` constructors and assignment operators [changematrix.cons]

```
explicit change_matrix(const matrix_2d& m) noexcept;
```

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

2 The matrix shall be set to the value of `m`.

8.14.3 `change_matrix` modifiers [changematrix.modifiers]

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

1 *Effects:* The matrix shall be set to the value of `m`.

8.14.4 `change_matrix` observers [changematrix.observers]

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

1 *Returns:* The value of the matrix.

8.15 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 8.1.

2 It has an origin of type `vector_2d`.

8.15.1 `change_origin` synopsis [changeorigin.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  namespace path_data {
    class change_origin {
    public:
      // 8.15.2, construct:
      explicit change_origin(const vector_2d& pt) noexcept;
```

```
    // 8.15.3, modifiers:
    void origin(const vector_2d& pt) noexcept;

    // 8.15.4, observers:
    vector_2d origin() const noexcept;
  };
};
} } } }
```

8.15.2 `change_origin` constructors and assignment operators [changeorigin.cons]

```
explicit change_origin(const vector_2d& pt) noexcept;
```

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

2 The origin shall be set to the value of `pt`.

8.15.3 `change_origin` modifiers [changeorigin.modifiers]

```
void origin(const vector_2d& value) noexcept;
```

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

8.15.4 `change_origin` observers [changeorigin.observers]

```
vector_2d origin() const noexcept;
```

1 *Returns:* The value of the origin.

8.16 Class `path_group` [pathgroup]

1 The class `path_group` contains the result of processing 8.1 a `path_factory` 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_factory` object.

8.16.1 `path_group` synopsis [pathgroup.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class path_group {
    public:
    // 8.16.2, construct/copy/destroy:
    explicit path_group(const path_factory& pb);
    path_group(const path_factory& pb, error_code& ec) noexcept;
  };
} } } }
```

8.16.2 `path_group` constructors [pathgroup.cons]

```
explicit path_group(const path_factory& pb);
path_group(const path_factory& 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_factory` object as specified at 8.1.

2 *Throws:* As specified in Error reporting (3).

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_factory`, 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_factory`, 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.

8.17 Class `path_factory` [pathfactory]

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

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

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

4 In addition to its path group, a path factory 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 factory 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{ }`.

8.17.1 `path_factory` synopsis [pathfactory.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  template <Allocator = allocator<path_data::path_data_types>>
  class path_factory {
    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>;

    // 8.17.3, construct, copy, move, destroy:
    path_factory() noexcept(noexcept(Allocator())) :
      path_factory(Allocator()) { }
```

```
explicit path_factory(const Allocator&) noexcept;
explicit path_factory(size_type n, const Allocator& = Allocator());
path_factory(size_type n, const value_type& value,
  const Allocator& = Allocator());
template <class InputIterator>
path_factory(InputIterator first, InputIterator last,
  const Allocator& = Allocator());
path_factory(const path_factory& x);
path_factory(path_factory&&) noexcept;
path_factory(const path_factory&, const Allocator&);
path_factory(path_factory&&, const Allocator&);
path_factory(initializer_list<value_type>, const Allocator& = Allocator());
~path_factory();
path_factory& operator=(const path_factory& x);
path_factory& operator=(path_factory&& x)
  noexcept(
  allocator_traits<Allocator>::propagate_on_container_move_assignment::value
  ||
  allocator_traits<Allocator>::is_always_equal::value);
path_factory& 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;

// 8.17.6, 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;

// 8.17.4, 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;

// 8.17.5, 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_factory&)
  noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value
    || allocator_traits<Allocator>::is_always_equal::value);
void clear() noexcept;

// 8.17.7, 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_factory<Allocator>& lhs,
    const path_factory<Allocator>& rhs);
  template <class Allocator>
  bool operator!=(const path_factory<Allocator>& lhs,
    const path_factory<Allocator>& rhs);

  // 8.17.8, specialized algorithms:
  template <Allocator>
  void swap(path_factory<Allocator>& lhs, path_factory<Allocator>& rhs)
    noexcept(noexcept(lhs.swap(rhs)));
} } } }
```

8.17.2 `path_factory` container requirements [pathfactory.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.

8.17.3 `path_factory` constructors, copy, and assignment [pathfactory.cons]

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

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

2 *Complexity:* Constant.

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

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

4 *Complexity:* Linear in `n`.

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

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

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

7 *Complexity:* Linear in `n`.

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

8 *Effects:* Constructs a `path_factory` 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.

8.17.4 `path_factory` capacity [pathfactory.capacity]

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

1 *Returns:* The total number of elements that the path factory 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 factory 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.

```
void swap(path_factory&)
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);
```

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

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.

8.17.5 `path_factory` modifiers [pathfactory.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 and destroys the value, if any, of the current point.

```
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 and sets the value of the current point to the value of the last-move-to point.

```
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.

5 If the current point does not contain a value, the last-move-to point shall be set to `vector_2d{ radius * cos(angle1), -(radius * -sin(angle1)) } + center`.

6 The current point shall be set to `vector_2d{ radius * cos(angle2), -(radius * -sin(angle2)) } + center`.

```
void arc_counterclockwise(const vector_2d& center, double radius,
  double angle1, double angle2) noexcept;
```

7 *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.

8 If the current point does not contain a value, the last-move-to point shall be set to `vector_2d{ radius * cos(angle2), radius * -sin(angle2) } + center`.

9 The current point shall be set to `vector_2d{ radius * cos(angle1), radius * -sin(angle1) } + center`.

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

10 *Effects:* If the current point does not contain a value, adds a `path_data::path_data_types` object constructed from `path_data::abs_move(pt0)` to the end of the path group.

11 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.

12 The current point shall be set to the value of `pt2`.

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

13 *Effects:* If the current point does not contain a value, the last-move-to point shall be set to the value of `pt`.

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

15 The current point shall be set to the value of `pt`.

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

16 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::abs_move(pt)` to the end of the path group.

17 The current point shall be set to the value of `pt`. The last-move-to point shall then be set to the value of the current point.

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

18 *Effects:* If the current point does not contain a value, adds a `path_data::path_data_types` object constructed from `path_data::abs_move(pt0)` to the end of the path group.

19 Adds a `path_data::path_data_types` object constructed from `path_data::abs_quadratic_curve(pt0, pt1)` to the end of the path group.

20 The current point shall be set to the value of `pt1`.

```
void rectangle(const experimental::io2d::rectangle& r) noexcept;
```

21 *Effects:*

1. Adds a `path_data::path_data_types` object constructed from `path_data::abs_move({ r.x(), r.y() })` to the end of the path group.
2. Adds a `path_data::path_data_types` object constructed from `path_data::rel_line({ r.width(), 0.0 })` to the end of the path group.
3. Adds a `path_data::path_data_types` object constructed from `path_data::rel_line({ 0.0, r.height() })` to the end of the path group.
4. Adds a `path_data::path_data_types` object constructed from `path_data::rel_line({ -r.width(), 0.0 })` to the end of the path group.
5. Adds a `path_data::path_data_types` object constructed from `path_data::close_path()` to the end of the path group.
6. The current point shall be set to `vector_2d{ r.x() r.y() }`.
7. The last-move-to point shall be set to `vector_2d{ r.x() r.y() }`.

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

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

23 *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.

24 The current point shall be set to the value of the current point added to `dpt2`.

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

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

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

27 The current point shall be set to the value of the current point added to `dpt`.

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

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

29 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::rel_move(dpt)` to the end of the path group.

30 The current point shall be set to the value of the current point added to `dpt`.The last-move-to point shall then be set to the value of the current point.

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

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

32 *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.

33 The current point shall be set to the value of the current point added to `dpt1`.

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

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

35 *Effects:* Adds a `path_data::path_data_types` object constructed from `(path_data::change_matrix(m)` to the end of the path group.

36 The transformation matrix shall be set to the value of `m`.

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

37 *Effects:* Adds a `path_data::path_data_types` object constructed from `path_data::change_origin(pt)` to the end of the path group.

38 The origin shall be set to the value of `pt`.

```
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);
```

39 *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.

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

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

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

42 *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 factory after the erased elements.

43 *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;
```

44 *Postconditions:* The current point shall not contain a value.

46 The transformation matrix shall have a value of `matrix_2d::init_identity{ }`.

47 The origin shall have a value of `vector_2d{ }`.

48 *Remarks:* The postconditions listed above are in addition to sequence container requirements for this function.

8.17.6 `path_factory` iterators [pathfactory.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 8.1 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 8.1 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 8.1 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 8.1 unless all of the violations are fixed prior to such processing.

8.17.7 `path_factory` observers [pathfactory.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 8.1. [*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.

8.17.8 `path_factory` specialized algorithms [pathfactory.special]

swap path_factory

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

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

9 Brushes [brushes]

1 Brushes serve as sources of visual data for composing operations.

There are four types of brushes:

(1.1) — color;

(1.2) — linear gradient;

(1.3) — radial gradient; and,

(1.4) — surface.

2 A brush has its own coordinate space (by default the standard coordinate space). It is controlled by a `matrix_2d` value.

3 It possesses a `filter` value and a `tiling` value, which combine to determine the visual data value returned when a composing operation samples a brush.

4 The `filter` value determines the value returned by a brush when a composing operation samples a brush. When the visual data of the brush is a pixmap and the point the composing operation requests does not directly correspond to the exact coordinate of a pixel within the pixmap, the `filter` value is used to determine the value of the visual data that is returned. If the visual data of the brush is not a pixmap, the `filter` value is irrelevant.

5 The `tiling` value controls what happens when a composing operation needs to sample from a point that is outside of the bounds of the brush.

6 Color brushes produce the same sampling result regardless of their coordinate space, `filter`, and `tiling`. They are unbounded and as such always produce the `rgba_color` value they were created with when sampled from, regardless of the requested point.

7 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 below (9.1).

8 Surface brushes take an `image_surface` object and use it as the source of the brush's visual data.

9.1 Gradient brushes [gradients]

1 Gradients have a `color_stop_group` as part of their observable state.

2 The `color_stop` objects within them contribute to defining a brush which, when sampled from, returns a value that is interpolated based on those color stops.

9.1.1 Linear gradients [gradients.linear]

1 A linear gradient is a type of gradient.

2 In addition to the observable state of a gradient, a linear gradient also has a *begin point* and an *end point* as parts of its observable state, both 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 `brush` object created using a degenerate linear gradient shall return the color `rgba_color::transparent_black()`. The remainder of 9.1.1 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 $(0, 1)$ 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.

9.1.2 Radial gradients [gradients.radial]

1 A radial gradient is a type of gradient.

2 In addition to the observable state of a gradient, a radial gradient also has a *start circle* and an *end circle* as part of its observable state, each of which is defined by a `vector_2d` object that denotes its center and a `double` value that denotes its radius.

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 shall return the color `rgba_color::transparent_black()`. The remainder of 9.1.2 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 `tiling` value of the `brush` object created using the radial gradient:

(8.1) — For `tiling::none`, the range of potential values for o is $[0, 1]$.

(8.2) — For all other `tiling` 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.

9.1.3 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 `rgba_color::transparent_black()` and the remainder of 9.1.3 is 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 9.1.3 is 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 9.1.3 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 `rgba_color::transparent_black()` and the remainder of 9.1.3 is 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 `rgba_color::transparent_black()` and the remainder of 9.1.3 is 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 `tiling` value of the brush which is created using the gradient:

(1.1) — If the `tiling` value is `tiling::none`, the color value of o shall be `rgba_color::transparent_black()`.

(1.2) — If the `tiling` value is `tiling::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 the `tiling` value is `tiling::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 the `tiling` value is `tiling::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, is performed within an RGB color space (with or without gamma correction), 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.

9.2 Enum class `tiling` [tiling]

9.2.1 `tiling` Summary [tiling.summary]

1 The `tiling` enum class describes how a point's visual data shall be determined if it is outside the bounds of the Source Brush (10.11.5) 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 `tiling` value with two exceptions:

a) If the point to be sampled is within the bounds of the Source Brush and the Source Brush's `tiling` value is `tiling::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 `tiling` value is `tiling::pad` rather than `tiling::none`.

b) If the point to be sampled is within the bounds of the Source Brush and the Source Brush's `tiling` value is `tiling::none`, ce Brush and the Source Brush's `tiling` value is `tiling::none`, then if the Source Brush's `filter` value requires that one or more points which are inside of the bounds of the Source Brush shall be sampled, each of those points shall be sampled such that the visual data that is returned shall be the equivalent of `rgba_color::transparent_black()`.

3 If a point to be sampled does not have a defined visual data element and the search for the nearest point with defined visual data produces two or more points with defined visual data that are equidistant from the point to be sampled, the returned visual data shall be an unspecified value which is the visual data of one of those equidistant points. Where possible, implementations should choose the among the equidistant points that have an x axisvalue and a y axisvalue that is nearest to `0.0`.

4 See Table 2 for the meaning of each `tiling` enumerator.

9.2.2 `tiling` Synopsis [tiling.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class tiling {
    none,
    repeat,
    reflect,
    pad
  };
} } } }
```

9.2.3 `tiling` Enumerators [tiling.enumerators]

Table 2 — `tiling` enumerator meanings

Enumerator	Meaning
`none`	If the point to be sampled is outside of the bounds of the Source Brush, the visual data that is returned shall be the equivalent of `rgba_color::transparent_black()`.
`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.

9.3 Enum class `filter` [filter]

9.3.1 `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 The result of sampling from a `brush` object `b` constructed from a `solid_color_brush_factory` is the same regardless of what filter is used and, as such, in these circumstances implementations should disregard the filter specified by the result of calling `b.filter()` when sampling from `b` and instead use an unspecified filter, even if that filter does not correspond to a filter specified by one of the `filter` enumerators.

4 See Table 3 for the meaning of each `filter` enumerator.

9.3.2 `filter` Synopsis [filter.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class filter {
    fast,
    good,
    best,
    nearest,
    bilinear
  };
} } } }
```

9.3.3 `filter` Enumerators [filter.enumerators]

Table 3 — `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.

9.4 Enum class `brush_type` [brushtype]

9.4.1 `brush_type` Summary [brushtype.summary]

1 The `brush_type` enum class denotes which brush factory was used to form a `brush` object.

2 See Table 4 for the meaning of each `brush_type` enumerator.

9.4.2 `brush_type` Synopsis [brushtype.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class brush_type {
    solid_color,
    surface,
    linear,
    radial
  };
} } } }
```

9.4.3 `brush_type` Enumerators [brushtype.enumerators]

Table 4 — `brush_type` enumerator meanings

Enumerator	Meaning
`solid_color`	The `brush` object was created from a `solid_color_brush_factory` object.
`surface`	The `brush` object was created from a `surface_brush_factory` object.
`linear`	The `brush` object was created from a `linear_brush_factory` object.
`radial`	The `brush` object was created from a `radial_brush_factory` object.

9.5 Color stops [colorstops]

1 A `color_stop_group` is a collection of zero or more `color_stop` objects which determine the values obtained by sampling a gradient (9.1) `brush`.

9.5.1 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 `rgba_color`.

9.5.1.1 `color_stop` Synopsis [colorstops.colorstop.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class color_stop {
  public:
        // 9.5.1.2, construct:
    constexpr color_stop(double o, const rgba_color& c);

    // 9.5.1.3, modifiers:
    void offset(double val) noexcept;
        void color(const rgba_color& val) noexcept;

    // 9.5.1.4, observers:
        double offset() const noexcept;
        rgba_color color() const noexcept;
  };
} } } }
```

9.5.1.2 `color_stop` constructors [colorstops.colorstop.cons]

```
constexpr color_stop(double o, const rgba_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`.

9.5.1.3 `color_stop` modifiers [colorstops.colorstop.modifiers]

```
void offset(double val) noexcept;
```

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

```
void color(double val) noexcept;
```

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

9.5.1.4 `color_stop` observers [colorstops.colorstop.observers]

```
double offset() const noexcept;
```

1 *Returns:* The value of the offset.

```
rgba_color color() const noexcept;
```

2 *Returns:* The value of the color.

9.5.2 Class `color_stop_group` [colorstops.colorstopgroup]

9.5.2.1 `color_stop_group` Synopsis [colorstops.colorstopgroup.synopsis]

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

    // 9.5.4, constructors:
    color_stop_group() noexcept(noexcept(Allocator())) :
    color_stop_group(Allocator()) { }
    explicit color_stop_group(const Allocator&) noexcept;
    explicit color_stop_group(size_type n, const Allocator& = Allocator());
    color_stop_group(size_type n, const value_type& value,
      const Allocator& = Allocator());
    template <class InputIterator>
    color_stop_group(InputIterator first, InputIterator last,
    const Allocator& = Allocator());
    color_stop_group(const color_stop_group& x);
    color_stop_group(color_stop_group&&) noexcept;
    color_stop_group(const color_stop_group&, const Allocator&);
    color_stop_group(color_stop_group&&, const Allocator&);
    color_stop_group(initializer_list<value_type>,
      const Allocator& = Allocator());
    ~color_stop_group();
    color_stop_group& operator=(const color_stop_group& x);
    color_stop_group& operator=(color_stop_group&& x)
      noexcept(
      allocator_traits<Allocator>::propagate_on_container_move_assignment::value
      || allocator_traits<Allocator>::is_always_equal::value);
    color_stop_group& operator=(initializer_list<value_type>);
```

```
template <class InputIterator>
void assign(InputIterator first, InputIterator last);
void assign(size_type n, const value_type& u);
void assign(initializer_list<value_type>);
allocator_type get_allocator() const noexcept;

// 9.5.7, iterators:
iterator begin() noexcept;
const_iterator begin() const noexcept;
const_iterator cbegin() const noexcept;

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

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

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

// 9.5.5, capacity
bool empty() const noexcept;
size_type size() const noexcept;
size_type max_size() const noexcept;
size_type capacity() const noexcept;
void resize(size_type sz);
void resize(size_type sz, const value_type& c);
void reserve(size_type n);
void shrink_to_fit();

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

// 9.5.6, modifiers:
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(color_stop_group&)
      noexcept(allocator_traits<Allocator>::propagate_on_container_swap::value
      || allocator_traits<Allocator>::is_always_equal::value);
    void clear() noexcept;
  };

  // 9.5.8, specialized algorithms:
  template <Allocator>
  void swap(color_stop_group<Allocator>& lhs, color_stop_group<Allocator>& rhs)
    noexcept(noexcept(lhs.swap(rhs)));
} } } }
```

9.5.3 `color_stop_group` container requirements [colorstops.colorstopgroup.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.

9.5.4 `color_stop_group` constructors, copy, and assignment [colorstops.colorstopgroup.cons]

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

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

2 *Complexity:* Constant.

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

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

4 *Complexity:* Linear in `n`.

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

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

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

7 *Complexity:* Linear in `n`.

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

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

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

9.5.5 `color_stop_group` capacity [colorstops.colorstopgroup.capacity]

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

1 *Returns:* The total number of elements that the color stop group 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 color stop group 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()`.[2]

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.

```
void swap(color_stop_group&)
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);
```

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

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

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

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

9.5.6 `color_stop_group` modifiers [colorstops.colorstopgroup.modifiers]

```
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);
```

1 *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.

2 *Complexity:* The complexity is linear in the number of elements inserted plus the distance to the end of the color stop group.

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

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

4 *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 vector after the erased elements.

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

9.5.7 `color_stop_group` iterators [colorstops.colorstopgroup.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 8.1 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.

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

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

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

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

9.5.8 `color_stop_group` specialized algorithms [colorstops.colorstopgroup.special]

swap color_stop_group

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

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

9.6 Class `brush` [brush]

9.6.1 `brush` Description [brush.intro]

1 The class `brush` describes an opaque wrapper for a graphics data graphics resource.

2 A `brush` object shall be 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 (10.11.5) or as a Mask Brush (10.11.4 and 10.11.10).

4 A `brush` object also has a tiling value of type `tiling`, a filter value of type `filter`, and a transformation matrix of type `matrix_2d`.

5 A `brush` object has an immutable `brush_type` observable state data value which indicates which type of brush it is (Table 4).

6 The `brush` object's graphics data may have less precision than the graphics data of the brush factory object from which it was created.

7 [*Example:* Several graphics and rendering technologies that are currently widely used store individual color and alpha channel data as 8-bit unsigned normalized integer values while the `double` type that is used by the `rgba_color` class for individual color and alpha is often a 64-bit value. It is precisely these situations which the preceding paragraph is intended to address. — *end example*]

9.6.2 `brush` synopsis [brush.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class brush {
  public:
    // 9.6.4, construct/copy/move/destroy:
```

```
    brush() = delete;
    brush(const brush&);
    brush& operator=(const brush&);
    brush(brush&& other) noexcept;
    brush& operator=(brush&& other) noexcept;

    brush(const rbga_color& c);
    brush(const rbga_color& c, error_code& ec) noexcept;
    template <Allocator = allocator<color_stop>>
    brush(const vector_2d& begin, const vector_2d& end,
      const color_stop_group<Allocator>& csg);
    template <Allocator = allocator<color_stop>>
    brush(const vector_2d& begin, const vector_2d& end,
      const color_stop_group<Allocator>& csg, error_code& ec) noexcept;
    template <Allocator = allocator<color_stop>>
    brush(const circle& start, const circle& end,
      const color_stop_group<Allocator>& csg);
    template <Allocator = allocator<color_stop>>
    brush(const circle& start, const circle& end,
      const color_stop_group<Allocator>& csg, error_code& ec) noexcept;
    brush(image_surface&& img);
    brush(image_surface&& img, error_code& ec) noexcept;

    // 9.6.5, modifiers:
    void tiling(experimental::io2d::tiling e) noexcept;
    void filter(experimental::io2d::filter f) noexcept;
    void matrix(const matrix_2d& m) noexcept;

    // 9.6.6, observers:
    experimental::io2d::tiling tiling() const noexcept;
    experimental::io2d::filter filter() const noexcept;
    matrix_2d matrix() const noexcept;
    brush_type type() const noexcept;
    const image_surface& surface() const;
    const image_surface& surface(error_code) const noexcept;
  };
} } } }
```

9.6.3 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*.

9.6.3.1 Sampling from a color brush [brush.sampling.color]

[1] When `b` is a color brush, then when sampling from `b`, the visual data returned shall always be the visual data equivalent `rgba_code` which was passed in when `b` was created, regardless of the point which is to be sampled and regardless of the return values of `b.tiling()`, `b.filter()`, and `b.matrix()`.

9.6.3.2 Sampling from a linear gradient brush [brush.sampling.linear]

1 When `b` is a linear gradient brush, then when sampling from `b`, the visual data returned shall be from the point `pt` in the rendered linear gradient, where `pt` is the return value when passing the point to be sampled to `b.matrix().transform_coords` and the rendered linear gradient is created as specified by 9.1.1 and 9.1.3, taking into account the value of `b.tiling()`.

9.6.3.3 Sampling from a radial gradient brush [brush.sampling.radial]

1 When `b` is a radial gradient brush, then when sampling from `b`, the visual data returned shall be from the point `pt` in the rendered radial gradient, where `pt` is the return value when passing the point to be sampled to `b.matrix().transform_coords` and the rendered radial gradient is created as specified by 9.1.2 and 9.1.3, taking into account the value of `b.tiling()`.

9.6.3.4 Sampling from a surface brush [brush.sampling.surface]

1 When `b` is a surface brush, then when sampling from `b`, the visual data returned shall be from the point `pt` in the graphics data of the brush, where `pt` is the return value when passing the point to be sampled to `b.matrix().transform_coords`, taking into account the value of `b.tiling()` and `b.filter()`.

9.6.4 `brush` constructors and assignment operators [brush.cons]

```
brush(const rbga_color& c);
brush(const rbga_color& c, error_code& ec) noexcept;
```

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

2 The brush shall be a color brush.

3 The brush's brush type shall be set to the value `brush_type::solid_color`.

4 The brush's tiling shall be set to the value `experimental::io2d::tiling::none`.

5 The brush's filter shall be set to the value `experimental::io2d::filter::fast`.

6 The brush's transformation matrix shall be set to the value `matrix_2d::init_identity()`.

7 The graphics data of the brush shall be created from the return value of `f.color()`. The visual data format of the graphics data shall be as if it is that specified by `format::argb`.

8 *Remarks:* Sampling from this brush shall produce the results specified in 9.6.3.1.

9 *Throws:* As specified in Error reporting (3).

10 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

`io2d_error::invalid_status` if there was a failure to allocate a resource other than memory.

```
template <Allocator = allocator<color_stop>>
brush(const vector_2d& begin, const vector_2d& end,
  const color_stop_group<Allocator>& csg);
template <Allocator = allocator<color_stop>>
brush(const vector_2d& begin, const vector_2d& end,
  const color_stop_group<Allocator>& csg, error_code& ec) noexcept;
```

11 *Effects:* Constructs an object of type `brush`.

12 The brush shall be a linear gradient brush.

13 The brush's brush type shall be set to the value `brush_type::linear`.

14 The brush's tiling shall be set to the value `experimental::io2d::tiling::none`.

15 The brush's filter shall be set to the value `experimental::io2d::filter::fast`.

16 The brush's transformation matrix shall be set to the value `matrix_2d::init_identity()`.

17 The graphics data of the brush is nominally as specified the introductory paragraphs of 9.1 and in 9.1.1. Its color stops shall be the values contained in `csg`. However because the graphics data is not directly observable, it is unspecified what data is stored and how it is stored, provided that the results of sampling from the brush are the same as if the brush's graphics data was stored as specified in the introductory paragraphs of 9.1 and in 9.1.1.

18 *Remarks:* Sampling from this brush shall produce the results specified in 9.6.3.2.

19 *Throws:* As specified in Error reporting (3).

20 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

`io2d_error::invalid_status` if there was a failure to allocate a resource other than memory.

```
template <Allocator = allocator<color_stop>>
brush(const circle& start, const circle& end,
  const color_stop_group<Allocator>& csg);
template <Allocator = allocator<color_stop>>
brush(const circle& start, const circle& end,
  const color_stop_group<Allocator>& csg, error_code& ec) noexcept;
```

21 *Effects:* Constructs an object of type `brush`.

22 The brush shall be a radial gradient brush.

23 The brush's brush type shall be set to the value `brush_type::radial`.

24 The brush's tiling shall be set to the value `experimental::io2d::tiling::none`.

25 The brush's filter shall be set to the value `experimental::io2d::filter::fast`.

26 The brush's transformation matrix shall be set to the value `matrix_2d::init_identity()`.

27 The graphics data of the brush is nominally as specified the introductory paragraphs of 9.1 and in 9.1.2. Its color stops shall be the values contained in `csg`. However because the graphics data is not directly observable, it is unspecified what data is stored and how it is stored, provided that the results of sampling from the brush are the same as if the brush's graphics data was stored as specified in the introductory paragraphs of 9.1 and in 9.1.2.

28 *Remarks:* Sampling from this brush shall produce the results specified in 9.6.3.3.

29 *Throws:* As specified in Error reporting (3).

30 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

`io2d_error::invalid_status` if there was a failure to allocate a resource other than memory.

```
brush(image_surface&& img);
brush(image_surface&& img, error_code& ec) noexcept;
```

31 *Effects:* Constructs an object of type `brush`.

33 The brush shall be a surface brush.

34 The brush's brush type shall be set to the value `brush_type::surface`.

35 The brush's tiling shall be set to the value `experimental::io2d::tiling::none`.

36 The brush's filter shall be set to the value `experimental::io2d::filter::good`.

37 The brush's transformation matrix shall be set to the value `matrix_2d::init_identity()`.

38 The graphics data of the brush shall be the underlying raster graphics data graphics resource of `img`.

39 *Remarks:* Sampling from this brush shall produce the results specified in 9.6.3.4.

40 *Throws:* As specified in Error reporting (3).

41 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

`io2d_error::invalid_status` if there was a failure to allocate a resource other than memory.

9.6.5 `brush` modifiers [brush.modifiers]

```
void tiling(experimental::io2d::tiling e) noexcept;
```

1 *Effects:* The brush's tiling shall be set to the value of `e`.

```
void filter(experimental::io2d::filter f) noexcept;
```

2 *Effects:* The brush's filter shall be set to the value `f`.

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

3 *Effects:* The brush's transformation matrix shall be set to the value `m`.

9.6.6 `brush` observers [brush.observers]

```
experimental::io2d::tiling tiling() const noexcept;
```

1 *Returns:* The value of brush's tiling.

```
experimental::io2d::filter filter() const noexcept;
```

2 *Returns:* The value of the brush's filter.

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

3 *Returns:* The value of the brush's transformation matrix.

```
brush_type type() const noexcept;
```

4 *Returns:* The brush's brush type.

```
  const image_surface& surface() const noexcept;
```

5 *Requires:* `*this.type() == brush_type::surface`.

6 *Returns:* A constant reference to the surface brush's `image_surface`.

10 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 a 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 (10.11.4).

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.

10.1 Enum class `antialias` [antialias]

10.1.1 `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 5 for the meaning of each `antialias` enumerator.

10.1.2 `antialias` Synopsis [antialias.synopsis]

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

10.1.3 `antialias` Enumerators [antialias.enumerators]

Table 5 — `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.

Table 5 — `antialias` enumerator meanings (continued)

Enumerator	Meaning
`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.

10.2 Enum class `content` [content]

10.2.1 `content` Summary [content.summary]

1 The `content` enum class describes the type of data that a `surface` object contains. See Table 6 for the meaning of each enumerator.

10.2.2 `content` Synopsis [content.synopsis]

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

10.2.3 `content` Enumerators [content.enumerators]

Table 6 — `content` value meanings

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

10.3 Enum class `fill_rule` [fillrule]

10.3.1 `fill_rule` Summary [fillrule.summary]

1 The `fill_rule` enum class determines how the Filling operation (10.11.8) 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.

10.3.2 `fill_rule` Synopsis [fillrule.synopsis]

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

10.3.3 `fill_rule` Enumerators [fillrule.enumerators]

Table 7 — `fill_rule` enumerator meanings

Enumerator	Meaning
`winding`	If the Fill Type (Table 16) is `fill_type::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 Type is `fill_type::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*]

10.4 Enum class `line_cap` [linecap]

10.4.1 `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.

10.4.2 `line_cap` Synopsis [linecap.synopsis]

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

10.4.3 `line_cap` Enumerators [linecap.enumerators]

Table 8 — `line_cap` enumerator meanings

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

10.5 Enum class `line_join` [linejoin]

10.5.1 `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.

10.5.2 `line_join` Synopsis [linejoin.synopsis]

```
namespace std { namespace experimental { namespace drawing { inline namespace
v1 {
  enum class line_join {
    miter,
    round,
    bevel
  };
} } } }
```

10.5.3 `line_join` Enumerators [linejoin.enumerators]

Table 9 — `line_join` enumerator meanings

Enumerator	Meaning
`miter`	Joins will be mitered or beveled, depending on the current Miter Limit (10.11.3).
`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.

10.6 Enum class `compositing_operator` [compositing.operator]

10.6.1 `compositing_operator` Summary [compositing.operator.summary]

[1] The `compositing_operator` enum class specifies composition algorithms. See Table 10, Table 11 and Table 12 for the meaning of each `compositing_operator` enumerator.

10.6.2 `compositing_operator` Synopsis [compositing.operator.synopsis]

```
namespace std { namespace experimental { namespace drawing { inline namespace
v1 {
  enum class compositing_operator {
    // 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
  };
} } } }
```

10.6.3 `compositing_operator` Enumerators [compositing.operator.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_operator` basic enumerator meanings

Enumerator	Bound	Color	Alpha
`clear`	Yes	$Rc = 0$	$Ra = 0$
`source`	Yes	$Rc = Sc$	$Ra = Sa$

Table 10 — `compositing_operator` basic enumerator meanings (continued)

Enumerator	Bound	Color	Alpha
`over`	N/A	$Rc = \frac{(Sca + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da \times (1 - Sa)$
`in`	No	$Rc = Sc$	$Ra = Sa \times Da$
`out`	No	$Rc = Sc$	$Ra = Sa \times (1 - Da)$
`atop`	N/A	$Rc = Sca + Dc \times (1 - Sa)$	$Ra = Da$
`dest`	N/A	$Rc = Dc$	$Ra = Da$
`dest_over`	N/A	$Rc = \frac{(Sca \times (1 - Da) + Dca)}{Ra}$	$Ra = (1 - Da) \times Sa + Da$
`dest_in`	No	$Rc = Dc$	$Ra = Sa \times Da$
`dest_out`	N/A	$Rc = Dc$	$Ra = (1 - Sa) \times Da$
`dest_atop`	No	$Rc = Sc \times (1 - Da) + Dca$	$Ra = Sa$
`xor_op`	N/A	$Rc = \frac{(Sca \times (1 - Da) + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da - 2 \times Sa \times Da$
`add`	N/A	$Rc = \frac{(Sca + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$
`saturate`	N/A	$Rc = \frac{(min(Sa, 1 - Da) \times Sc + Dca)}{Ra}$	$Ra = min(1, Sa + Da)$

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

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

21 The color formula divides 1 by the result color's alpha channel value. As a result, if the result color's alpha channel is zero then a division by zero would normally occur. Implementations shall not throw an exception nor otherwise produce any observable error condition if the result color's alpha channel is zero. Instead, implementations shall bypass the division by zero and produce the result color (0.0, 0.0, 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_operator` blend enumerator meanings

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

`overlay`	$if(Dc \leq 0.5)$ {
	$f(Sc, Dc) = 2 \times Sc \times Dc$
	}
	$else$ {
	$f(Sc, Dc) =$
	$1 - 2 \times (1 - Sc) \times$
	$(1 - Dc)$
	}
	[*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_operator` blend enumerator meanings (continued)

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

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

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

}
return C
}

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_operator` hsl enumerator meanings

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

10.7 Enum class `format` [format]

10.7.1 `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*]

10.7.2 `format` Synopsis [format.synopsis]

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

10.7.3 `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.

10.8 Enum class `scaling` [scaling]

10.8.1 `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 (10.13.1) differs from the size of its Back Buffer (10.13.1).

2 See Table 14 for the meaning of each `scaling` enumerator.

10.8.2 `scaling` Synopsis [scaling.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class scaling {
    letterbox,
    uniform,
    fill_uniform,
    fill_exact,
    none
  };
} } } }
```

10.8.3 `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 (10.13.1) of `ds` is 640x480 (i.e. it has a width of 640 pixels and a height of 480 pixels), giving it an aspect ratio of $1.\bar{3}$.

The Display Buffer (10.13.1) of `ds` is 1280x720, giving it an aspect ratio of $1.\bar{7}$.

When a rectangle is defined in an example, the coordinate $(x1, y1)$ denotes the top left corner of the rectangle, inclusive, and the coordinate $(x2, y2)$ denotes the bottom right corner of the rectangle, exclusive. As such, a rectangle with $(x1, y1) = (10, 10)$, $(x2, y2) = (20, 20)$ is 10 pixels wide and 10 pixels tall and includes the pixel $(x, y) = (19, 19)$ but does not include the pixels $(x, y) = (20, 19)$ or $(x, y) = (19, 20)$. — *end note*]

Table 14 — `scaling` enumerator meanings

Enumerator	Meaning
`letterbox`	Fill the Display Buffer with the Letterbox Brush (10.13.4) 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*]

10.9 Enum class `refresh_rate` [refreshrate]

10.9.1 `refresh_rate` Summary [refreshrate.summary]

1 The `refresh_rate` enum class describes when the Draw Callback (Table 19) of a `display_surface` object shall be called. See Table 15 for the meaning of each enumerator.

10.9.2 `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
  };
} } } }
```

10.9.3 `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 19) 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*]

10.10 Class `device` [device]

10.10.1 `device` synopsis [device.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class device {
  public:
    // See 2.3
    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);

    // 10.10.3, modifiers:
    void flush() noexcept;
    void lock();
    void lock(error_code& ec) noexcept;
```

```
    void unlock();
    void unlock(error_code& ec) noexcept;
  };
} } } }
```

10.10.2 `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.

10.10.3 `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 (3).

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 (3).

17 *Remarks:* This function shall not be called more times than `lock` has been called; no diagnostic is required.

10.11 Class `surface` [surface]

10.11.1 `surface` description [surface.intro]

1 The `surface` class provides an interface for managing a graphics data graphics resource and its observable state data (10.11.3).

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 10.11.4 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.

10.11.2 `surface` synopsis [surface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class surface {
  public:
    // 10.11.11, constructors, assignment operators, destructors:
    surface() = delete;
    surface(const surface&) = delete;
    surface& operator=(const surface&) = delete;
    surface(surface&& other) noexcept;
    surface& operator=(surface&& other) noexcept;
    virtual ~surface();

    // 10.11.12, state modifiers:
    void finish() noexcept;
```

```
void flush();
void flush(error_code& ec) noexcept;
shared_ptr<experimental::io2d::device> device();
shared_ptr<experimental::io2d::device> device(error_code& ec) noexcept;
void mark_dirty();
void mark_dirty(error_code& ec) noexcept;
void mark_dirty(const rectangle& rect);
void mark_dirty(const rectangle& rect, error_code& ec) noexcept;
void map(const function<void(mapped_surface&)>& action);
void map(const function<void(mapped_surface&, error_code&)>& action,
  error_code& ec);
void map(const function<void(mapped_surface&)>& action,
  const rectangle& extents);
void map(const function<void(mapped_surface&, error_code&)>& action,
  const rectangle& extents, error_code& ec);
void push_state();
void push_state(error_code& ec) noexcept;
void pop_state();
void pop_state(error_code& ec) noexcept;
void brush(nullvalue_t) noexcept;
void brush(const experimental::io2d::brush& source);
void brush(const experimental::io2d::brush& source, error_code& ec)
  noexcept;
void antialias(experimental::io2d::antialias a) noexcept;
void dashes(experimental::nullvalue_t) noexcept;
void dashes(const experimental::io2d::dashes& d);
void dashes(const experimental::io2d::dashes& d, error_code& ec) noexcept;
void fill_rule(experimental::io2d::fill_rule fr) noexcept;
void line_cap(experimental::io2d::line_cap lc) noexcept;
void line_join(experimental::io2d::line_join lj) noexcept;
void line_width(double width) noexcept;
void miter_limit(double limit) noexcept;
void compositing_operator(experimental::io2d::compositing_operator co)
  noexcept;
void clip(const experimental::io2d::path& p);
void clip(const experimental::io2d::path& p, error_code& ec) noexcept;
void clip_immediate();
void clip_immediate(error_code& ec) noexcept;
void path(experimental::nullvalue_t) noexcept;
void path(const experimental::io2d::path& p);
void path(const experimental::io2d::path& p, error_code& ec) noexcept;

// 10.11.13, immediate path modifiers:
experimental::io2d::path_factory& immediate() noexcept;

// 10.11.14, render modifiers:
void fill();
void fill(error_code& ec) noexcept;
void fill(const rgba_color& c);
void fill(const rgba_color& c, error_code& ec) noexcept;
void fill(const experimental::io2d::brush& b);
void fill(const experimental::io2d::brush& b, error_code& ec) noexcept;
void fill(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
```

```
void fill(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
void fill_immediate();
void fill_immediate(error_code& ec) noexcept;
void fill_immediate(const rgba_color& c);
void fill_immediate(const rgba_color& c, error_code& ec) noexcept;
void fill_immediate(const experimental::io2d::brush& b);
void fill_immediate(const experimental::io2d::brush& b, error_code& ec)
  noexcept;
void fill_immediate(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void fill_immediate(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
void paint();
void paint(error_code& ec) noexcept;
void paint(const rgba_color& c);
void paint(const rgba_color& c, error_code& ec) noexcept;
void paint(const experimental::io2d::brush& b);
void paint(const experimental::io2d::brush& b, error_code& ec) noexcept;
void paint(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void paint(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
void paint(double alpha);
void paint(double alpha, error_code& ec) noexcept;
void paint(const rgba_color& c, double alpha);
void paint(const rgba_color& c, double alpha, error_code& ec) noexcept;
void paint(const experimental::io2d::brush& b, double alpha);
void paint(const experimental::io2d::brush& b, double alpha,
  error_code& ec) noexcept;
void paint(const surface& s, double alpha,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void paint(const surface& s, double alpha, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good)
  noexcept;
void stroke();
void stroke(error_code& ec) noexcept;
void stroke(const rgba_color& c);
void stroke(const rgba_color& c, error_code& ec) noexcept;
void stroke(const experimental::io2d::brush& b);
void stroke(const experimental::io2d::brush& b, error_code& ec) noexcept;
void stroke(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void stroke(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
void stroke_immediate();
```

```
void stroke_immediate(error_code& ec) noexcept;
void stroke_immediate(const rgba_color& c);
void stroke_immediate(const rgba_color& c, error_code& ec) noexcept;
void stroke_immediate(const experimental::io2d::brush& b);
void stroke_immediate(const experimental::io2d::brush& b, error_code& ec)
  noexcept;
void stroke_immediate(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void stroke_immediate(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;

// 10.11.15, mask render modifiers:
void mask(const experimental::io2d::brush& mb);
void mask(const experimental::io2d::brush& mb, error_code& ec)
  noexcept;
void mask(const experimental::io2d::brush& mb, const rgba_color& c);
void mask(const experimental::io2d::brush& mb, const rgba_color& c,
  error_code& ec) noexcept;
void mask(const experimental::io2d::brush& mb,
  const experimental::io2d::brush& b);
void mask(const experimental::io2d::brush& mb,
  const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask(const experimental::io2d::brush& mb, const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void mask(const experimental::io2d::brush& mb, const surface& s,
  error_code& ec, matrix m = matrix_2d::init_identity(),
  tiling e = tiling::none, filter f = filter::good) noexcept;
void mask(surface& ms,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask(surface& ms, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask(surface& ms, const rgba_color& c,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask(surface& ms, const rgba_color& c, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask(surface& ms, const experimental::io2d::brush& b,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask(surface& ms, const experimental::io2d::brush& b, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask(surface& ms, const surface& s,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, tiling e = tiling::none,
  filter msFilter = filter::good, filter f = filter::good);
void mask(surface& ms, const surface& s, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
```

```
    const matrix_2d& m = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, tiling e = tiling::none,
    filter msFilter = filter::good, filter f = filter::good) noexcept;
  void mask_immediate(const experimental::io2d::brush& mb);
  void mask_immediate(const experimental::io2d::brush& mb, error_code& ec)
    noexcept;
  void mask_immediate(const experimental::io2d::brush& mb,
    const rgba_color& c);
  void mask_immediate(const experimental::io2d::brush& mb,
    const rgba_color& c, error_code& ec) noexcept;
  void mask_immediate(const experimental::io2d::brush& mb,
    const experimental::io2d::brush& b);
  void mask_immediate(const experimental::io2d::brush& mb,
    const experimental::io2d::brush& b, error_code& ec) noexcept;
  void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
    const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
    filter f = filter::good);
  void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
    const matrix_2d& m = matrix_2d::init_identity(), error_code& ec,
    tiling e = tiling::none, filter f = filter::good) noexcept;
  void mask_immediate(surface& ms,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, filter msFilter = filter::good);
  void mask_immediate(surface& ms, error_code& ec,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
  void mask_immediate(surface& ms, const rgba_color& c,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, filter msFilter = filter::good);
  void mask_immediate(surface& ms, const rgba_color& c, error_code& ec,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
  void mask_immediate(surface& ms, const experimental::io2d::brush& b,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, filter msFilter = filter::good);
  void mask_immediate(surface& ms, const experimental::io2d::brush& b,
    error_code& ec, const matrix_2d& msMatrix = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
  void mask_immediate(surface& ms, const surface& s,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    const matrix_2d& m = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, tiling e = tiling::none,
    filter msFilter = filter::good, filter f = filter::good);
  void mask_immediate(surface& ms, const surface& s, error_code& ec,
    const matrix_2d& msMatrix = matrix_2d::init_identity(),
    const matrix_2d& m = matrix_2d::init_identity(),
    tiling msTiling = tiling::none, tiling e = tiling::none,
    filter msFilter = filter::good, filter f = filter::good) noexcept;

  // 10.11.16, transformation modifiers:
  void matrix(const matrix_2d& matrix);
  void matrix(const matrix_2d& matrix, error_code& ec) noexcept;

  // 10.11.17, state observers:
  bool is_finished() const noexcept;
```

```
    experimental::io2d::content content() const noexcept;
    experimental::io2d::brush brush() const noexcept;
    experimental::io2d::antialias antialias() const noexcept;
    experimental::io2d::dashes dashes() const;
    experimental::io2d::dashes dashes(error_code& ec) const noexcept;
    experimental::io2d::fill_rule fill_rule() const noexcept;
    experimental::io2d::line_cap line_cap() const noexcept;
    experimental::io2d::line_join line_join() const noexcept;
    double line_width() const noexcept;
    double miter_limit() const noexcept;
    experimental::io2d::compositing_operator compositing_operator() const
      noexcept;
    rectangle clip_extents() const noexcept;
    bool in_clip(const vector_2d& pt) const noexcept;
    vector<rectangle> clip_rectangles() const;
    vector<rectangle> clip_rectangles(error_code& ec) const noexcept;

    // 10.11.18, render observers:
    rectangle fill_extents() const noexcept;
    rectangle fill_extents_immediate() const;
    rectangle fill_extents_immediate(error_code& ec) const noexcept;
    bool in_fill(const vector_2d& pt) const noexcept;
    bool in_fill_immediate(const vector_2d& pt) const;
    bool in_fill_immediate(const vector_2d& pt, error_code& ec) const noexcept;
    rectangle stroke_extents() const noexcept;
    rectangle stroke_extents_immediate() const;
    rectangle stroke_extents_immediate(error_code& ec) const noexcept;
    bool in_stroke(const vector_2d& pt) const noexcept;
    bool in_stroke_immediate(const vector_2d& pt) const;
    bool in_stroke_immediate(const vector_2d& pt, error_code& ec) const
      noexcept;

    // 10.11.19, transformation observers:
    matrix_2d matrix() const noexcept;
    vector_2d user_to_surface(const vector_2d& pt) const noexcept;
    vector_2d user_to_surface_distance(const vector_2d& dpt) const noexcept;
    vector_2d surface_to_user(const vector_2d& pt) const noexcept;
    vector_2d surface_to_user_distance(const vector_2d& dpt) const noexcept;
  };
} } } }
```

10.11.3 `surface` state [surface.state]

1 Table 16 specifies the name, type, function, and default value for each item of a surface's observable state.

10.11.3.1 `surface` state default values [surface.state.default]

Table 16 — Surface observable state

Name	Type	Function	Default value
Current Brush	`brush`	This is the brush that shall be available for use when performing rendering and composing operations	`brush{ { rgba_color::black() } }`

Table 16 — Surface observable state (continued)

Name	Type	Function	Default value
General Antialiasing	`antialias`	This is the type of antialiasing that should be used when performing non-text rendering operations	`antialias::default_antialias`
Dash Pattern	`dashes`	This specifies the pattern that shall be used when performing stroke rendering operations	`dashes{ vector<double>(), 0.0 }`
Fill Rule	`fill_rule`	This controls which areas of paths shall be eligible to be filled when performing fill rendering operations	`fill_rule::winding`
Line Cap	`line_cap`	This specifies how the end of a path segment that is not joined to another path segment shall be rendered when performing stroke rendering operations	`line_cap::butt`
Line Join	`line_join`	This specifies how the join point of two path segments that are joined to each other shall be rendered when performing stroke rendering operations	`line_join::miter`
Line Width	`double`	This is the width, in surface coordinate space units, that shall be used to determine the rendered width of path segments when performing a stroke rendering operation	`2.0`
Miter Limit	`double`	This is the value that shall be used to calculate whether a line join shall be mitered or beveled when the value of Line Join is `line_join::miter`.	`10.0`
Composition Operator	`compositing_operator`	This specifies the composition algorithm that shall be used when performing rendering operations	`compositing_operator::over`
Clip Area	*`unspecified`*	The area or areas of the underlying graphics data graphics resource which are the only areas in which composing operations can have any effect. It is in the Surface Coordinate Space (10.11.6)	An unbounded area

Table 16 — Surface observable state (continued)

Name	Type	Function	Default value
Current Path	`path_group`	This is the `path_group` object that shall be used when performing non-immediate rendering operations. It is in the User Coordinate Space (10.11.6)	`path{ path_factory{ } }`
Immediate Path	`path_factory`	This is the `path_factory` object that shall be used when performing immediate rendering operations. It is in the User Coordinate Space (10.11.6)	`path_factory{ }`
Transformation Matrix	`matrix_2d`	This is the `matrix_2d` object that shall be used to transform points to and from the Surface Coordinate Space (10.11.6)	`matrix_2d::init_identity{ }`

1 [*Note:* The Clip Area may be modified by intersecting the current Clip Area with the areas of a `path_group` object that would be filled according to the current Fill Rule. The resulting Clip Area will only contain areas that were already in the Clip Area and does not need to be a valid `path_group` object since the Clip Area is only guaranteed to be observable by the effects it has on composing operations. For this reason, the Clip Area's type is unspecified. The Clip Area may also be modified by resetting it to its default value, which is the only way of enlarging the current Clip Area. — *end note*]

10.11.3.2 `surface` saved state [surface.state.save]

1 A surface object provides an interface to save its current state and subsequently restore it using `surface::push_state` and `pop_state`, respectively.

2 Each call to `surface::pop_state` restores a surface object's state to its values at the time of the most recent call to `surface::push_state`.

3 The following list denotes each item of observable state that shall be saved by `surface::push_state` and restored by `surface::pop_state`, using the state item names listed in Table 16:

(3.1) — Current Brush

(3.2) — General Antialiasing

(3.3) — Dash Pattern

(3.4) — Fill Rule

(3.5) — Line Cap

(3.6) — Line Join

(3.7) — Line Width

(3.8) — Miter Limit

(3.9) — Compositing Operator

(3.10) — Tolerance

(3.11) — Clip Area

(3.12) — Current Path

(3.13) — Immediate Path

(3.14) — Transformation Matrix

10.11.4 `surface` rendering and composing operations [surface.rendering]

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`, `surface::fill_immediate`
Stroking	`surface::stroke`, `surface::stroke_immediate`
Masking	`surface::mask`, `surface::mask_immediate`

2 The filling, stroking, and masking operations each provide two functions, not including overloads, for invoking their functionality.

3 Certain rendering and composing operations require a *Source Path*, which is a path group.

4 The rendering and composing operations invoked by the `surface::fill`, `surface::stroke`, and `surface::mask` functions shall use the `surface` object's Current Path as their Source Path.

5 The rendering and composing operations invoked by the `surface::fill_immediate`, `surface::stroke_immediate`, and `surface::mask_immediate` functions shall use as their Source Path a `path_group` formed in the manner specified by 8.1 from the surface's Immediate Path.

6 Provided that none of the function calls results in an error, given a `surface` object `s`, there shall be no observable difference in the results of:

(6.1) — calling `surface::fill_immediate` on `s` versus calling `surface::fill` on `s` with the same arguments immediately after creating a `path_group` object from the result of calling `surface::immediate` on `s` and then immediately setting that `path_group` object as the Current Path by calling `surface::path` on `s` using that `path_group` object as the argument to that function;

(6.2) — calling `surface::stroke_immediate` on `s` versus calling `surface::stroke` on `s` with the same arguments immediately after creating a `path_group` object from the result of calling `surface::immediate` on `s` and then immediately setting that `path_group` object as the Current Path by calling `surface::path` on `s` using that `path_group` object as the argument to that function;

(6.3) — calling `surface::mask_immediate` on `s` versus calling `surface::mask` on `s` with the same arguments immediately after creating a `path_group` object from the result of calling `surface::immediate` on `s` and then immediately setting that `path_group` object as the Current Path by calling `surface::path` on `s` using that `path_group` object as the argument to that function.

7 The painting operation is a single-part operation consisting of a composing operation. Each of the other operations is a multi-part operation consisting of a rendering operation followed by a composing operation.

8 The surface's Clip Area shall always apply to all composing operations. This is true even if no mention is made of the Clip Area in the description of a composing operation or if the description of a composing

operation uses phraseology that could otherwise be read as being unconstrained. [*Note:* Because of this, mention of the applicability of the Clip Area will be omitted in the descriptions of the composing operation portions of the rendering and composing operations that follow. — *end note*]

9 Each of the four fundamental rendering and composing operations has its own set of minimum arguments.

1. The Painting operation has no minimum arguments.
2. The Filling operation has no minimum arguments.
3. The Stroking operation has no minimum arguments.
4. The Masking operation has a Mask Brush (10.11.10) which can be composed of one or four minimum arguments: if the first argument to the Masking operation is of type `brush`, it has one minimum argument (that `brush` object); otherwise the first argument is of type `surface` and other three arguments are the first `matrix_2d` argument, the first `tiling` argument, and the first `filter` argument, resulting in four minimum arguments.

10 If a reference to an `error_code` object (see 3) is passed to a rendering and composing operation, it shall be considered part of the set of minimum arguments for that operation.

11 For the Masking operation, where a `surface` object is used as the Mask Brush argument, it may be immediately followed by an argument of type `vector_2d` which shall be used as the origin of the `surface` object used as the Mask Brush (see 10.11.10) and it shall be considered part of the set of minimum arguments for that operation.

10.11.5 Source Brush for rendering and composing operations [surface.sourcebrush]

1 A *Source Brush* is composed of a graphics data graphics resource, an `tiling` value, a `filter` value, and a `matrix_2d` object.

2 A `brush` object provides all of the components of a Source Brush. As such, `brush` objects will commonly be used as Source Brushes.

3 The rendering and composing operations require a Source Brush each time they are invoked. The following paragraphs specify how the Source Brush shall be determined.

4 A Source Brush can be provided as a `brush` object or as an identified set of arguments which, together, provide all of the components of a Source Brush.

5 When a rendering and composing operation is invoked with only those arguments that are part of its set of minimum arguments, the `surface` object's Current Brush shall be the Source Brush.

6 When a rendering and composing operation is invoked with *extra arguments*, which are additional arguments beyond those that are part of its set of minimum arguments, the Source Brush shall be as follows, with any errors reported as specified in Error reporting (3):

(6.1) — Where there is one extra argument and it is a `brush` object, the Source Brush shall be that `brush` object.

(6.2) — Where there is one extra argument and it is an `rgba_color` object, the Source Brush shall be the same as if it were a `brush` object constructed from a `solid_color_brush_factory` constructed from the extra argument and sampling from it shall be as specified by 9.6.3 and 9.6.3.1.

(6.3) — Where there is more than one extra argument, including default arguments, and the first extra argument is a `surface` object, the Source Brush shall be the same as if it were a `brush` object `b` constructed from a `surface_brush_factory` constructed from the `surface` object extra argument and sampling from it shall be as specified by 9.6.3 and 9.6.3.4, subject to the following adjustments:

1. The `surface` object extra argument should be used directly; no copy of it should be made. This differs from the `surface_brush_factory` class, which requires that a copy of the surface be made and that only the copy be used.
2. The Source Brush shall produce the same observable results that `b` would produce after `brush::tiling` was called on `b` with the `tiling` object extra argument passed to `brush::tiling` as an argument.
3. The Source Brush shall produce the same observable results that `b` would produce after `brush::filter` was called on `b` with the `filter` object extra argument passed to `brush::filter` as an argument.
4. The Source Brush shall produce the same observable results that `b` would produce after `brush::matrix` was called on `b` with the `matrix` object extra argument passed to `brush::matrix` as an argument.

7 [*Note:* The rendering and composing operations each have the same combinations of extra arguments, all of which are covered by the above requirements that determine the Source Brush when extra arguments are present. — *end note*]

8 A `surface` object shall not be used as a Mask Brush or as a Source Brush for any rendering and composing operations invoked on that `surface` object; no diagnostic is required. [*Note:* This restriction does not apply to surface brushes since they are not `surface` objects. — *end note*]

10.11.6 Standard coordinate spaces [surface.coordinatespaces]

1 There are four standard coordinate spaces relevant to the rendering and composing operations (10.11.4):

(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 (10.11.5). Its transformation matrix is as if it was a copy of the Source Brush's `matrix_2d` object.

3 The *Mask Coordinate Space* is the standard coordinate space of the Mask Brush (10.11.10). Its transformation matrix is as if it was a copy of the Mask Brush's `matrix_2d` object.

4 The *User Coordinate Space* is the standard coordinate space of `surface` object. Its transformation matrix is the return value of `matrix_2d::init_identity`.

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 return value of calling `s.matrix()` where `s` is the `surface` object.

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 18 — Point transformations

From	To	Transform
Brush Coordinate Space	Mask Coordinate Space	`mcsm.transform_point(bcsm.invert().transform_point(pt))`.
Brush Coordinate Space	User Coordinate Space	`bcsm.invert().transform_point(pt)`.

Table 18 — Point transformations (continued)

From	To	Transform
Brush Coordinate Space	Surface Coordinate Space	`scsm.transform_point(bcsm.invert().transform_point(pt))`.
User Coordinate Space	Brush Coordinate Space	`bcsm.transform_point(pt)`.
User Coordinate Space	Mask Coordinate Space	`mcsm.transform_point(pt)`.
User Coordinate Space	Surface Coordinate Space	`scsm.transform_point(pt)`.
Surface Coordinate Space	Brush Coordinate Space	`bcsm.transform_point(scsm.invert().transform_point(pt))`.
Surface Coordinate Space	Mask Coordinate Space	`mcsm.transform_point(scsm.invert().transform_point(pt))`.
Surface Coordinate Space	User Coordinate Space	`scsm.invert().transform_point(pt)`.

10.11.7 `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 (Table 16); if so, proceed with the remaining steps.
2. Transform *sp* from the Surface Coordinate Space (10.11.6) to the Brush Coordinate Space (Table 18), resulting in point *bp*.
3. Sample from point *bp* of the Source Brush (10.11.5), 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 (Table 16), 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.

10.11.8 `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 (Table 16); if so, proceed with the remaining steps.
2. Transform *sp* from the Surface Coordinate Space (10.11.6) to the User Coordinate Space (Table 18), resulting in point *up*.
3. Using the Source Path (10.11.4) and the Fill Rule (Table 16), 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 (10.11.6 and Table 18), resulting in point *bp*.
5. Sample from point *bp* of the Source Brush (10.11.5), 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 (Table 16), 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.

10.11.9 `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 (10.11.4).

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 (10.11.6) to the Surface Coordinate Space (10.11.6). [*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::butt`, 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*]

10.11.10 `surface` masking [surface.masking]

1 The Masking operation uses a *Mask Brush* as described in 10.11.4.

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 (Table 16); if so, proceed with the remaining steps.
2. Transform *sp* from the Surface Coordinate Space (10.11.6) to the Mask Coordinate Space (Table 18), 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 (10.11.5), 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 (Table 16), 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.

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

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

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

10.11.12 `surface` state modifiers [surface.modifiers.state]

```
void finish() noexcept;
```

1 *Effects:* Releases all resources managed by the `surface` object `s` if `!s.is_finished()`, otherwise does nothing.

2 *Postconditions:* `s.is_finished()` shall return `true`.

3 *Remarks:* Once this function has been called, the surface is *finished*. The only valid operations on a finished surface are destruction, calling `finish()`, and calling `is_finished()`. Except as otherwise noted, any other operation on a finished surface or any attempt to use a finished `surface` or `surface`-derived object as an argument to a function produces undefined behavior; no diagnostic is required.

4 This function is intended to release the resources of both the `surface` class and of any class derived from it. As such, it should call a private or protected virtual member function in order to guarantee correct behavior.

```
void flush();
void flush(error_code& ec) noexcept;
```

5 *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.

6 If the implementation does provide a native handle to the `surface` object's underlying graphics data graphics resource, then:

(6.1) — Any pending rendering and composing operations (10.11.4) shall be performed on the `surface` object. [*Note:* As long as the observable effect is the same as if they were performed immediately, it does not matter whether they are executed, batched, or otherwise committed to the underlying rendering and presentation technologies. — *end note*]

(6.2) — The implementation may then modify the `surface` object's observable state (10.11.3).

(6.3) — The `surface` object's observable state shall then be undefined.

7 *Throws:* As specified in Error reporting (3).

8 *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.

9 *Error conditions:* The potential errors are implementation-defined.

10 Implementations should avoid producing errors here.

11 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.

12 *Notes:* Because the `surface` object's observable state can change as a result of calling this function, users will typically want to call `surface::push_state` before calling this function. When the user is done using the `surface` object's native handle, he or she must call `surface::mark_dirty` if changes were made to the underlying graphics data graphics resource using the native handle as per that function's semantics. Once that is done, or immediately after access using the native handle is finished if no changes were made, the user needs to then call `surface::pop_state` (assuming a previous, valid call to surface::push_state). Otherwise the user must set every item of observable state before using the `surface` object in any other way.

```
shared_ptr<experimental::io2d::device> device();
shared_ptr<experimental::io2d::device> device(error_code& ec) noexcept;
```

13 *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.

14 *Throws:* As specified in Error reporting (3).

15 *Error conditions:* `errc::not_enough_memory` if a `device` object needs to be created and not enough memory exists to do so.

```
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;
```

16 *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.

17 If the implementation does provide a native handle to the `surface` object's underlying graphics data graphics resource, then:

(17.1) — 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.

(17.2) — 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.

18 *Throws:* As specified in Error reporting (3).

19 *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.

20 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*]

21 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

22 If the implementation does not provide a native handle to the `surface` object's underlying graphics data graphics resource, this function shall not produce any errors.

```
void map(const function<void(mapped_surface&)>& action);
void map(const function<void(mapped_surface&, error_code&)>& action, error_code& ec);
void map(const function<void(mapped_surface&)>& action, const rectangle& extents);
void map(const function<void(mapped_surface&, error_code&)>& action,
  const rectangle& extents, error_code& ec);
```

23 *Effects:* Creates a `mapped_surface` object and calls `action` using it.

24 The `mapped_surface` object is created using `*this`, which allows direct manipulation of the underlying graphics data graphics resource.

25 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.

26 *Throws:* As specified in Error reporting (3).

27 *Remarks:* Whether changes are committed to the underlying graphics data graphics resource immediately or only when the `mapped_surface` object is destroyed is unspecified.

28 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.

29 *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.

```
void push_state();
void push_state(error_code& ec) noexcept;
```

30 *Effects:* Saves the state specified in 10.11.3.2 as if to an internal stack of saved states.

31 *Remarks:* This function is intended to save the state of both the `surface` class and of any class derived from it. As such, it should call a private or protected virtual member function in order to guarantee correct behavior.

32 *Throws:* As specified in Error reporting (3).

33 *Error conditions:* `errc::not_enough_memory` if the state cannot be saved.

```
void pop_state();
void pop_state(error_code& ec) noexcept;
```

34 *Effects:* Restores the state of `*this` to the values saved by the most recent call to `surface::push_state`.

35 *Throws:* As specified in Error reporting (3).

36 *Remarks:* This function is intended to restore the state of both the `surface` class and of any class derived from it. As such, it should call a private or protected virtual member function in order to guarantee correct behavior.

37 Because this function is only restoring previously saved state, except where the conditions for `io2d_error::invalid_pop_state` are met, implementations should not generate errors.

38 *Error conditions:* `io2d_error::invalid_pop_state` if this function is called without a previous matching call to `surface::push_state`. Implementations shall not produce `io2d_error::invalid_pop_state` except under the conditions stated in this paragraph.

39 Excluding the previously specified error, any errors produced by calling this function are implementation-defined.

```
void brush(nullvalue_t) noexcept;
```

40 *Effects:* Sets the Current Brush (Table 16) to be a `brush` object that is the same as if it was the brush that is the default value of Current Brush 10.11.3.1.

41 [*Note:* This function does not require that the Current Brush be set to the same `brush` object that was the original default value Current Brush `brush` object. That said, because this function is `noexcept`, implementers can (and likely should) keep a reference to the original, default value Current Brush `brush` object and use it when this function is called. Allocating a new `brush` object may result in errors due to memory constraints or other considerations. — *end note*]

```
void brush(const experimental::io2d::brush& source);
void brush(const experimental::io2d::brush& source,
  error_code& ec) noexcept;
```

42 *Effects:* Sets `source` as the Current Brush (Table 16), ensuring that it shall not be destroyed and shall be recreated, if necessary, by the underlying rendering and presentation technologies so that it shall be available for use at least until such time as it is no longer the Current Brush of any `surface` or `surface`-derived object.

43 *Throws:* As specified in Error reporting (3).

44 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void antialias(experimental::io2d::antialias a) noexcept;
```

45 *Effects:* Sets General Antialiasing (Table 16) to the value of `a`.

```
void dashes(nullvalue_t) noexcept;
```

46 *Effects:* Sets the Dash Pattern (Table 16) to be the default value of Dash Pattern.

```
void dashes(const experimental::io2d::dashes& d);
void dashes(const experimental::io2d::dashes& d, error_code& ec) noexcept;
```

47 *Effects:* Sets the Dash Pattern (Table 16) to the value of `d`.

48 *Throws:* As specified in Error reporting (3).

49 *Error conditions:* `errc::not_enough_memory` if there is a problem caching the new Dash Pattern.

50 `io2d_error::invalid_dash` if the new Dash Pattern contains a negative value or if it has values and all of them are `0.0`.

```
void fill_rule(experimental::io2d::fill_rule fr) noexcept;
```

51 *Effects:* Sets the Fill Rule (Table 16) to the value of `fr`.

```
void line_cap(experimental::io2d::line_cap lc) noexcept;
```

52 *Effects:* Sets the Line Cap (Table 16) to the value of `lc`.

```
void line_join(experimental::io2d::line_join lj) noexcept;
```

53 *Effects:* Sets the Line Join (Table 16) to the value of `lj`.

```
void line_width(double width) noexcept;
```

54 *Effects:* Sets the Line Width (Table 16) to `max(width, 0.0)`.

```
void miter_limit(double limit) noexcept;
```

55 *Effects:* Sets the Miter Limit (Table 16) to the value of `limit` clamped to be within the range defined by the implementation-defined minimum value and the implementation-defined maximum value, inclusive.

56 *Remarks:* The *implementation-defined*minimum value shall not be greater than `2.0` and the *implementation-defined*maximum value shall not be less than `10.0`.

57 *Notes:* The Miter Limit only applies when the Line Join is set to `line_join::miter`. — *end note*]

58 *Remarks:* The Miter Limit works as follow. The length of the miter is divided by the width of the line. If the resulting value is greater than the Miter Limit, the join will be beveled, otherwise it will be mitered.

```
void compositing_operator(experimental::io2d::compositing_operator co)
  noexcept;
```

59 *Effects:* Sets the Composition Operator to the value of `co`.

```
void clip(const experimental::io2d::path& p);
void clip(const experimental::io2d::path& p, error_code& ec) noexcept;
```

60 *Effects:* Sets the Clip Area (Table 16) to be the intersection of the Clip Area and `p` where `p`'s area is determined in the same way as if `p` were filled according to the Fill Rule.

61 *Notes:* The Clip Area never increases as a result of this function.

```
void clip_immediate();
void clip_immediate(error_code& ec) noexcept;
```

62 *Effects:* Sets the Clip Area (Table 16) to be the intersection of the Clip Area and the Immediate Path (Table 16) where the Immediate Path's area is determined in the same way as if the Immediate Path were filled according to the Fill Rule.

63 *Notes:* The Clip Area never increases as a result of this function.

```
void path(nullvalue_t) noexcept;
```

64 *Effects:* Set's the Current Path (Table 16).

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

```
void path(const experimental::io2d::path& p);
void path(const experimental::io2d::path& p, error_code& ec) noexcept;
```

66 *Effects:* Set's Current Path (Table 16) to `p`.

67 *Remarks:* Processing the path data so that it is properly transformed can be done at the time it is first set as a path on a `surface` object or any time before that. The untransformed `vector<path_data_item>` shall be retained to ensure that a path can be properly recreated at any time. The steps for converting an untransformed `vector<path_data_item>` to transformed path data are found at 8.1.

10.11.13 `surface` immediate path modifiers [surface.modifiers.immediatepath]

```
experimental::io2d::path_factory& immediate() noexcept;
```

1 *Returns:* A reference to the Immediate Path (Table 16).

10.11.14 `surface` render modifiers [surface.modifiers.render]

```
void fill();
void fill(error_code& ec) noexcept;
void fill(const rgba_color& c);
void fill(const rgba_color& c, error_code& ec) noexcept;
void fill(const experimental::io2d::brush& b);
void fill(const experimental::io2d::brush& b, error_code& ec) noexcept;
void fill(const surface& s, const matrix_2d& m = matrix_2d::init_identity(),
  tiling e = tiling::none, filter f = filter::good);
void fill(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
```

1 *Effects:* Performs the Filling rendering and composing operation as specified by 10.11.8.

2 The meanings of the parameters are specified by 10.11.4.

3 *Throws:* As specified in Error reporting (3).

4 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void fill_immediate();
void fill_immediate(error_code& ec) noexcept;
void fill_immediate(const rgba_color& c);
void fill_immediate(const rgba_color& c, error_code& ec) noexcept;
void fill_immediate(const experimental::io2d::brush& b);
void fill_immediate(const experimental::io2d::brush& b, error_code& ec)
  noexcept;
void fill_immediate(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling e = tiling::none, filter f = filter::good);
void fill_immediate(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling e = tiling::none, filter f = filter::good) noexcept;
```

5 *Effects:* Performs the Filling rendering and composing operation as specified by 10.11.8.

6 The meanings of the parameters are specified by 10.11.4.

7 *Throws:* As specified in Error reporting (3).

8 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void paint();
void paint(error_code& ec) noexcept;
void paint(const rgba_color& c);
void paint(const rgba_color& c, error_code& ec) noexcept;
void paint(const experimental::io2d::brush& b);
void paint(const experimental::io2d::brush& b, error_code& ec) noexcept;
void paint(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void paint(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
```

```
  filter f = filter::good) noexcept;
void paint(double alpha);
void paint(double alpha, error_code& ec) noexcept;
void paint(const rgba_color& c, double alpha);
void paint(const rgba_color& c, double alpha, error_code& ec) noexcept;
void paint(const experimental::io2d::brush& b, double alpha);
void paint(const experimental::io2d::brush& b, double alpha,
  error_code& ec) noexcept;
void paint(const surface& s, double alpha,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void paint(const surface& s, double alpha, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
```

9 *Effects:* Performs the Painting rendering and composing operation as specified by 10.11.7.

10 The meanings of the parameters are specified by 10.11.4.

11 *Throws:* As specified in Error reporting (3).

12 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void stroke();
void stroke(error_code& ec) noexcept;
void stroke(const rgba_color& c);
void stroke(const rgba_color& c, error_code& ec) noexcept;
void stroke(const experimental::io2d::brush& b);
void stroke(const experimental::io2d::brush& b, error_code& ec) noexcept;
void stroke(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void stroke(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
```

13 *Effects:* Performs the Stroking rendering and composing operation as specified by 10.11.9.

14 The meanings of the parameters are specified by 10.11.4.

15 *Throws:* As specified in Error reporting (3).

16 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void stroke_immediate();
void stroke_immediate(error_code& ec) noexcept;
void stroke_immediate(const rgba_color& c);
void stroke_immediate(const rgba_color& c, error_code& ec) noexcept;
void stroke_immediate(const experimental::io2d::brush& b);
void stroke_immediate(const experimental::io2d::brush& b, error_code& ec)
  noexcept;
void stroke_immediate(const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void stroke_immediate(const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
```

17 *Effects:* Performs the Stroking rendering and composing operation as specified by 10.11.9.

18 The meanings of the parameters are specified by 10.11.4.

19 *Throws:* As specified in Error reporting (3).

20 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

10.11.15 `surface` mask render modifiers [surface.modifiers.maskrender]

```
void mask(const experimental::io2d::brush& mb);
void mask(const experimental::io2d::brush& mb, error_code& ec)
  noexcept;
void mask(const experimental::io2d::brush& mb, const rgba_color& c);
void mask(const experimental::io2d::brush& mb, const rgba_color& c,
  error_code& ec) noexcept;
void mask(const experimental::io2d::brush& mb,
  const experimental::io2d::brush& b);
void mask(const experimental::io2d::brush& mb,
  const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask(const experimental::io2d::brush& mb, const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void mask(const experimental::io2d::brush& mb, const surface& s, error_code& ec,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good) noexcept;
void mask(surface& ms,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask(surface& ms, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask(surface& ms, const rgba_color& c,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask(surface& ms, const rgba_color& c, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask(surface& ms, const experimental::io2d::brush& b,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask(surface& ms, const experimental::io2d::brush& b, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask(surface& ms, const surface& s,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, tiling e = tiling::none,
  filter msFilter = filter::good, filter f = filter::good);
void mask(surface& ms, const surface& s, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, tiling e = tiling::none,
  filter msFilter = filter::good, filter f = filter::good) noexcept;
```

1 *Effects:* Performs the Masking rendering and composing operation as specified by 10.11.10.

2 The meanings of the parameters are specified by 10.11.4.

3 *Throws:* As specified in Error reporting (3).

4 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void mask_immediate(const experimental::io2d::brush& mb);
  void mask_immediate(const experimental::io2d::brush& mb,
  error_code& ec) noexcept;
void mask_immediate(const experimental::io2d::brush& mb,
  const rgba_color& c);
void mask_immediate(const experimental::io2d::brush& mb,
  const rgba_color& c, error_code& ec) noexcept;
void mask_immediate(const experimental::io2d::brush& mb,
  const experimental::io2d::brush& b);
void mask_immediate(const experimental::io2d::brush& mb,
  const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), tiling e = tiling::none,
  filter f = filter::good);
void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
  const matrix_2d& m = matrix_2d::init_identity(), error_code& ec,
  tiling e = tiling::none, filter f = filter::good) noexcept;
void mask_immediate(surface& ms,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask_immediate(surface& ms, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask_immediate(surface& ms, const rgba_color& c,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask_immediate(surface& ms, const rgba_color& c, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask_immediate(surface& ms, const experimental::io2d::brush& b,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good);
void mask_immediate(surface& ms, const experimental::io2d::brush& b,
  error_code& ec, const matrix_2d& msMatrix = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, filter msFilter = filter::good) noexcept;
void mask_immediate(surface& ms, const surface& s,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, tiling e = tiling::none,
  filter msFilter = filter::good, filter f = filter::good);
void mask_immediate(surface& ms, const surface& s, error_code& ec,
  const matrix_2d& msMatrix = matrix_2d::init_identity(),
  const matrix_2d& m = matrix_2d::init_identity(),
  tiling msTiling = tiling::none, tiling e = tiling::none,
  filter msFilter = filter::good, filter f = filter::good) noexcept;
```

5 *Effects:* Performs the Masking rendering and composing operation as specified by 10.11.10.

6 The meanings of the parameters are specified by 10.11.4.

7 *Throws:* As specified in Error reporting (3).

8 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

10.11.16 `surface` transformation modifiers [surface.modifiers.transform]

```
void matrix(const matrix_2d& m);
void matrix(const matrix_2d& m, error_code& ec) noexcept;
```

1 *Effects:* Sets Transformation Matrix (Table 16) to the value of `m`.

2 *Throws:* As specified in Error reporting (3).

3 *Error conditions:* `io2d_error::invalid_matrix` if calling `!m.is_invertible()`.

10.11.17 `surface` state observers [surface.observers.state]

```
bool is_finished() const noexcept;
```

1 *Returns:* If a call to `surface::finished` has previously been made, returns `true`; otherwise returns `false`.

```
experimental::io2d::content content() const noexcept;
```

2 *Returns:* The `content` enumerator (10.2.3) that appropriately describes the underlying graphics data graphics resource.

```
experimental::io2d::brush brush() const noexcept;
```

3 *Returns:* The Current Brush (Table 16).

```
experimental::io2d::antialias antialias() const noexcept;
```

4 *Returns:* The value of General Antialiasing (Table 16).

```
experimental::io2d::dashes dashes() const;
experimental::io2d::dashes dashes(error_code& ec) const noexcept;
```

5 *Returns:* The Dash Pattern (Table 16).

6 *Throws:* As specified in Error reporting (3).

7 *Remarks:* See 10.11.9 for more information about the `dashes` type.

8 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

```
experimental::io2d::fill_rule fill_rule() const noexcept;
```

9 *Returns:* The value of Fill Rule (Table 16).

```
experimental::io2d::line_cap line_cap() const noexcept;
```

10 *Returns:* The value of Line Cap (Table 16).

```
experimental::io2d::line_join line_join() const noexcept;
```

11 *Returns:* The value of Line Join (Table 16).

```
double line_width() const noexcept;
```

12 *Returns:* The value of Line Width (Table 16).

```
double miter_limit() const noexcept;
```

13 *Returns:* The value of Miter Limit (Table 16).

```
experimental::io2d::compositing_operator compositing_operator() const
  noexcept;
```

14 *Returns:* The value of Composition Operator (Table 16).

```
rectangle clip_extents() const noexcept;
```

15 *Returns:* A `rectangle` that specifies the smallest bounding box which contains the Clip Area (Table 16).

```
bool in_clip(const vector_2d& pt) const noexcept;
```

16 *Returns:* If the point `pt` is outside of the Clip Area (Table 16), returns `false`; otherwise returns `true`.

```
vector<rectangle> clip_rectangles() const;
vector<rectangle> clip_rectangles(error_code& ec) const noexcept;
```

17 *Returns:* A `vector<rectangle>` object which contains the rectangles which make up the Clip Area (Table 16).

18 If an `error_code&` argument is passed and the error `io2d_error::clip_not_representable` occurs, returns an empty `vector<rectangle>` object.

19 *Throws:* As specified in Error reporting (3).

20 *Error conditions:* `io2d_error::clip_not_representable` if the Clip Area contains one or more areas that cannot be represented using rectangles. [*Example:* This error would occur if the Clip Area contains arcs or curves. — *end example*]

21 `errc::not enough memory` if a failure to allocate memory occurs.

10.11.18 `surface` render observers [surface.observers.render]

```
rectangle fill_extents() const noexcept;
```

1 *Returns:* A `rectangle` that specifies the smallest bounding box which contains all areas in which a Filling operation (10.11.8) can have an effect using the Current Path (Table 16) and the value of Fill Rule. The Clip Area and the bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

2 For purposes of calculating the return value, the coordinates of the Current Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. When the final values for the return value are calculated, they shall be transformed using `surface::surface_to_user` and the `rectangle` that is returned shall be composed of those transformed values.

3 *Notes:* The resulting bounding box is not the same as the area that would be changed if `surface::fill` was called since it is calculated without regard to the Current Brush, the Composition Operator, and the Clip Area and thus could contain areas that would not actually be affected by a call to `surface::fill`.

```
rectangle fill_extents_immediate() const;
rectangle fill_extents_immediate(error_code& ec) const noexcept;
```

4 *Returns:* A `rectangle` that specifies the smallest bounding box which contains all areas in which a Filling operation (10.11.8) can have an effect using the Immediate Path (Table 16) and the value of Fill Rule. The Clip Area and the bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

5 For purposes of calculating the return value, the processed coordinates of the Immediate Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. When the final values for the return value are calculated, they shall be transformed using `surface::surface_to_user` and the `rectangle` that is returned shall be composed of those transformed values.

6 *Throws:* As specified in Error reporting (3).

7 *Remarks:* This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 8.1.

8 *Error conditions:* `io2d_error::no_current_point` if, when processing the Immediate Path's path group, an operation was encountered which required a current point and the current path had no current point.

9 `io2d_error::invalid_matrix` if, when processing the Immediate Path's path group, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

10 Other errors, if any, produced by this function are implementation-defined.

11 *Notes:* The resulting bounding box is not the same as the area that would be changed if `surface::fill_immediate` was called since it is calculated without regard to the Current Brush, the Composition Operator, and the Clip Area and thus could contain areas that would not actually be affected by a call to `surface::fill_immediate`.

```
bool in_fill(const vector_2d& pt) const noexcept;
```

12 *Returns:* If the point `pt` is not within any of the areas in which a Filling operation (10.11.8) can have an effect using the Current Path (Table 16) and the value of Fill Rule, this function returns `false`; otherwise it returns `true`. The Clip Area and bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

13 For purposes of calculating the return value, the coordinates of the Current Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. The value of `pt` shall also be transformed using `surface::user_to_surface` prior to determining whether or not it is within any of the Filling operation areas.

14 *Notes:* The result does not mean that the content at the point `pt` would be changed if `surface::fill` was called since the areas are calculated without regard to the Current Brush, the Composition Operator, the Clip Area, and the Transformation Matrix and thus could contain areas that would not actually be affected by a call to `surface::fill`.

```
bool in_fill_immediate(const vector_2d& pt) const;
bool in_fill_immediate(const vector_2d& pt, error_code& ec) const noexcept;
```

15 *Returns:* If the point `pt` is not within any of the areas in which a Filling operation (10.11.8) can have an effect using the Immediate Path (Table 16) and the value of Fill Rule, this function returns `false`; otherwise it returns `true`. The Transformation Matrix, Clip Area, and the bound of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

16 For purposes of calculating the return value, the processed coordinates of the Immediate Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. The value of `pt` shall also be transformed using `surface::user_to_surface` prior to determining whether or not it is within any of the Filling operation areas.

17 *Throws:* As specified in Error reporting (3).

18 *Remarks:* This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 8.1.

19 *Error conditions:* `io2d_error::no_current_point` if, when processing the Immediate Path's path group, an operation was encountered which required a current point and the current path had no current point.

20 `io2d_error::invalid_matrix` if, when processing the Immediate Path's path group, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

21 Other errors, if any, produced by this function are implementation-defined.

22 *Notes:* The result does not mean that the point `pt` would be changed if `surface::fill` was called since the areas are calculated without regard to the Current Brush, the Composition Operator, the Clip Area, and the Transformation Matrix and thus could contain areas that would not actually be affected by a call to `surface::fill`.

```
rectangle stroke_extents() const noexcept;
```

23 *Returns:* A `rectangle` that specifies the smallest bounding box which contains all areas in which a Stroking operation (10.11.9) can have an effect using the Current Path (Table 16), the Line Cap, the Line Join, the Line Width, the Miter Limit, and the Dash Pattern. The Clip Area and the bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

24 For purposes of calculating the return value, the coordinates of the Current Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. When the final values for the return value are calculated, they shall be transformed using `surface::surface_to_user` and the `rectangle` that is returned shall be composed of those transformed values.

25 *Notes:* The resulting bounding box is not the same as the area that would be changed if `surface::stroke` was called since it is calculated without regard to the Current Brush, the Composition Operator, and the Clip Area and thus could contain areas that would not actually be affected by a call to `surface::stroke`.

```
rectangle stroke_extents_immediate() const;
rectangle stroke_extents_immediate(error_code& ec) const noexcept;
```

26 *Returns:* A `rectangle` that specifies the smallest bounding box which contains all areas in which a Stroking operation (10.11.9) can have an effect using the Current Path (Table 16), the Line Cap, the Line Join, the Line Width, the Miter Limit, and the Dash Pattern. The Clip Area and the bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

27 For purposes of calculating the return value, the processed coordinates of the Immediate Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. When the final values for the return value are calculated, they shall be transformed using `surface::surface_to_user` and the `rectangle` that is returned shall be composed of those transformed values.

28 *Throws:* As specified in Error reporting (3).

29 *Remarks:* This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 8.1.

30 *Error conditions:* `io2d_error::no_current_point` if, when processing the Immediate Path's path group, an operation was encountered which required a current point and the current path had no current point.

31 `io2d_error::invalid_matrix` if, when processing the Immediate Path's path group, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

32 Other errors, if any, produced by this function are implementation-defined.

33 *Notes:* The resulting bounding box is not the same as the area that would be changed if `surface::stroke_immediate` was called since it is calculated without regard to the Current Brush, the Composition Operator, and the Clip Area and thus could contain areas that would not actually be affected by a call to `surface::stroke_immediate`.

```
bool in_stroke(const vector_2d& pt) const noexcept;
```

34 *Returns:* If the point `pt` is not within any of the areas in which a Stroking operation (10.11.9) can have an effect using the Current Path (Table 16), the Line Cap, the Line Join, the Line Width, the Miter Limit, and the Dash Pattern, this function returns `false`; otherwise it returns `true`. The Clip Area and bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

35 For purposes of calculating the return value, the coordinates of the Current Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. The value of `pt` shall also be transformed using `surface::user_to_surface` prior to determining whether or not it is within any of the Stroking operation areas.

36 *Notes:* The result does not mean that the content at the point `pt` would be changed if `surface::stroke` was called since the areas are calculated without regard to the Current Brush, the Composition Operator, and the Clip Area and thus could contain areas that would not actually be affected by a call to `surface::stroke`.

```
bool in_stroke_immediate(const vector_2d& pt) const;
bool in_stroke_immediate(const vector_2d& pt, error_code& ec) const noexcept;
```

37 *Returns:* If the point `pt` is not within any of the areas in which a Stroking operation (10.11.9) can have an effect using the Immediate Path (Table 16), the Line Cap, the Line Join, the Line Width, the Miter Limit, and the Dash Pattern, this function returns `false`; otherwise it returns `true`. The Clip Area and bounds of the underlying graphics data graphics resource shall be disregarded for purposes of calculating the return value.

38 For purposes of calculating the return value, the processed coordinates of the Immediate Path shall be transformed using `surface::user_to_surface` before any other calculations are performed. The value of `pt` shall also be transformed using `surface::user_to_surface` prior to determining whether or not it is within any of the Stroking operation areas.

39 *Throws:* As specified in Error reporting (3).

40 *Remarks:* This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 8.1.

41 *Error conditions:* `io2d_error::no_current_point` if, when processing the Immediate Path's path group, an operation was encountered which required a current point and the current path had no current point.

42 `io2d_error::invalid_matrix` if, when processing the Immediate Path's path group, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

43 Other errors, if any, produced by this function are implementation-defined.

44 *Notes:* The result does not mean that the content at the point `pt` would be changed if `surface::stroke_immediate` was called since the areas are calculated without regard to the Current Brush, the Composition Operator, and the Clip Area and thus could contain areas that would not actually be affected by a call to `surface::stroke_immediate`.

10.11.19 `surface` transformation observers [surface.observers.transform]

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

1 *Returns:* The Transformation Matrix (Table 16).

```
vector_2d user_to_surface(const vector_2d& pt) const noexcept;
```

2 *Returns:* The result of calling `matrix_2d::transform_point` on Transformation Matrix (Table 16) with `pt` as the argument to that function.

```
vector_2d user_to_surface_distance(const vector_2d& dpt) const noexcept;
```

3 *Returns:* The result of calling `matrix_2d::transform_distance` on Transformation Matrix (Table 16) with `dpt` as the argument to that function.

```
vector_2d surface_to_user(const vector_2d& pt) const noexcept;
```

4 *Returns:* The result of creating a copy of Transformation Matrix (Table 16), calling `matrix_2d::invert` on that copy, and then calling `matrix_2d::transform_point` on the copy with `pt` as the argument to that function.

```
vector_2d surface_to_user_distance(const vector_2d& dpt) const noexcept;
```

5 *Returns:* The result of creating a copy of Transformation Matrix (Table 16), calling `matrix_2d::invert` on that copy, and then calling `matrix_2d::transform_distance` on the copy with `dpt` as the argument to that function.

10.12 Class `image_surface` [imagesurface]

10.12.1 `image_surface` synopsis [imagesurface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class image_surface : public surface {
  public:
    // 10.12.3, construct/copy/move/destroy:
    image_surface() = delete;
    image_surface(const image_surface&) = delete;
    image_surface& operator=(const image_surface&) = delete;
    image_surface(image_surface&& other) noexcept;
    image_surface& operator=(image_surface&& other) noexcept;
    image_surface(experimental::io2d::format fmt, int width, int height);
    image_surface(experimental::io2d::format fmt, int width, int height,
      error_code& ec) noexcept;
    image_surface(vector<unsigned char>& data, experimental::io2d::format fmt,
      int width, int height);
    image_surface(vector<unsigned char>& data, experimental::io2d::format fmt,
      int width, int height, error_code& ec) noexcept;
    virtual ~image_surface();

    // 10.12.4, modifiers:
    void data(const vector<unsigned char>& data);
    void data(const vector<unsigned char>& data, error_code& ec) noexcept;
    vector<unsigned char> data();
    vector<unsigned char> data(error_code& ec) noexcept;

    // 10.12.5, observers:
    experimental::io2d::format format() const noexcept;
    int width() const noexcept;
    int height() const noexcept;
    int stride() const noexcept;
  };
} } } }
```

10.12.2 `image_surface` Description [imagesurface.intro]

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 it. — *end note*]

10.12.3 `image_surface` constructors and assignment operators [imagesurface.cons]

```
image_surface(experimental::io2d::format fmt, int width, int height);
image_surface(experimental::io2d::format fmt, int width, int height,
  error_code& ec) noexcept;
```

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

2 *Postconditions:* `this->format() == fmt`.

3 `this->width() == width`.

4 `this->height() == height`.

5 *Throws:* As specified in Error reporting (3).

6 *Remarks:* The result of calling `this->data()` shall be `0` for all bits that are defined by the specification of that function. [*Note:* Given implementation-specific details, it is possible that not all bits of the `image_surface` object's underlying raster graphics data graphics resource will be used to determine its pixel data. The values of those unused bits are irrelevant and the above paragraph makes it clear that only the bits that matter in determining pixel data have defined values, which are specified to have the same value as the bits of `data`; the value of the other bits, if any, do not have any defined value. — *end note*]

7 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
image_surface(vector<unsigned char>& data, experimental::io2d::format fmt,
  int width, int height);
image_surface(vector<unsigned char>& data, experimental::io2d::format fmt,
  int width, int height, error_code& ec) noexcept;
```

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

9 *Postconditions:* `this->format() == fmt`.

10 `this->width() == width`.

11 `this->height() == height`.

12 `this->data() == data` for all bits that are defined by the specification of that function. [*Note:* Given implementation-specific details, it is possible that not all bits of the `image_surface` object's underlying raster graphics data graphics resource will be used to determine its pixel data. The values of those unused bits are irrelevant and the above paragraph makes it clear that only the bits that matter in determining pixel data have defined values, which are specified to have the same value as the bits of `data`; the value of the other bits, if any, do not have any defined value. — *end note*]

13 *Throws:* As specified in Error reporting (3).

14 *Error conditions:* `io2d_error::invalid_stride` if `format_stride_for_width(fmt, width) * height != data.size()`.

15 Other errors, if any, produced by this function are implementation-defined.

```
virtual ~image_surface();
```

16 *Effects:* Destroys an object of type `image_surface`.

10.12.4 `image_surface` modifiers [imagesurface.modifiers]

```
void data(const vector<unsigned char>& data);
void data(const vector<unsigned char>& data, error_code& ec) noexcept;
```

1 *Effects:* Any pending rendering and composing operations (10.11.4) shall be performed.

2 *Postconditions:* `this->data() == data` for all bits that are defined by the specification of that function. [*Note:* Given implementation-specific details, it is possible that not all bits of the `image_surface` object's underlying raster graphics data graphics resource will be used to determine its pixel data. The values of those unused bits are irrelevant and the above paragraph makes it clear that only the bits that matter in determining pixel data have defined values, which are specified to have the same value as the bits of `data`; the value of the other bits, if any, do not have any defined value. — *end note*]

3 *Throws:* As specified in Error reporting (3).

4 *Error conditions:* `io2d_error::invalid_stride` if `format_stride_for_width(fmt, width) * height != data.size()`.

5 Other errors, if any, produced by this function are implementation-defined.

```
vector<unsigned char> data();
vector<unsigned char> data(error_code& ec) noexcept;
```

6 *Effects:* Any pending rendering and composing operations (10.11.4) shall be performed.

7 *Returns:* A `vector<unsigned char>` containing the byte values of the pixel data of the underlying raster graphics data graphics resource. Where the result of `this->format()` is a `format` value which denotes a multi-byte pixel format, the pixel data shall be in native-endian order.

8 *Throws:* As specified in Error reporting (3).

9 *Error conditions:* `errc::not_enough_memory` if there was a failure to allocate memory.

10 *Notes:* This would normally be an observer function but the requirement that "[a]ny pending rendering and composing operations (10.11.4) shall be performed" means that calling this function might modify the underlying raster graphics data graphics resource. As such this function cannot be marked `const` and thus cannot strictly be classified as an observer function.

11 Developers using this function are cautioned that in many graphics technologies that implementers might use to implement this functionality, the effects of this function will typically cause a large performance degradation and as such it should be used with care and avoided where possible.

10.12.5 `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::unknown || 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::unknown || this->format() == experimental::io2d::format::invalid`.

```
int stride() const noexcept;
```

7 *Returns:* The length, in bytes, of a horizontal line of the `image_surface` object. [*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*]

8 *Remarks:* This function shall return the value 0 if `this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid`.

10.13 Class `display_surface` [displaysurface]

10.13.1 `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 19) 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*]

10.13.2 `display_surface` synopsis [displaysurface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class display_surface : public surface {
  public:
    // 10.13.5, construct/copy/move/destroy:
    display_surface() = delete;
    display_surface(const display_surface&) = delete;
    display_surface& operator=(const display_surface&) = delete;
    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;

  virtual ~display_surface();

  // 10.13.6, 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(experimental::nullvalue_t) noexcept;
  void letterbox_brush(const rgba_color& c);
  void letterbox_brush(const rgba_color& c, error_code& ec) noexcept;
  void letterbox_brush(const experimental::io2d::brush& b);
  void letterbox_brush(const experimental::io2d::brush& b, error_code& ec)
    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 show();
  int show(error_code& ec) noexcept;
  void exit_show();
  void exit_show(error_code& ec) noexcept;
  void exit_show(std::chrono::duration d);
  void exit_show(std::chrono::duration d, error_code& ec) noexcept;
```

```
    // 10.13.7, 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;
    tuple<int, int> dimensions() const noexcept;
    tuple<int, int> 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;
    experimental::io2d::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;
  };
} } } }
```

10.13.3 `display_surface` miscellaneou behavior [displaysurface.misc]

1 What constitutes an output device is implementation-defined, with the sole constraint being that an output device shall 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 need not 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.

10.13.4 `display_surface` state [displaysurface.state]

1 Table 19 specifies the name, type, function, and default value for each item of a display surface's observable state.

2 Because the `display_surface` class publicly derives from the `surface` class, the observable state of a display surface also includes the observable state of a surface, as specified at 10.11.3.1.

Table 19 — 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{ { rgba_color::black() } }`
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`

Table 19 — Display surface observable state (continued)

Name	Type	Function	Default value
Draw Width	`int`	The width in pixels of the Back Buffer. The minimum value is `1`. The maximum value is unspecified. Because users can only request a preferred value for the Draw Width when setting and altering it, the maximum value may be a run-time determined value. If the preferred Draw Width exceeds the maximum value, then if a preferred Draw Height has also been supplied then implementations should provide a Back Buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred Draw Width and the preferred Draw Height otherwise implementations should provide a Back Buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred Draw Width and the current Draw Height	*N/A* [*Note:* It is impossible to create a `display_surface` object without providing a preferred Draw Width value; as such a default value cannot exist. — *end note*]

Table 19 — 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 19 — 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 19 — 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 19 — 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 19 — 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)	

10.13.5 `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 *Effects:* Constructs an object of type `display_surface`.

2 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.

3 The preferredFormat parameter specifies the preferred pixel format value for Draw Format.

4 The value of Scaling Type shall be the value of `scl`.

5 The value of Refresh Rate shall be the value of `rr`.

6 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.

7 All other observable state data shall have their default values.

8 *Throws:* As specified in Error reporting (3).

9 *Error conditions:* `errc::invalid_argument` if `preferredWidth <= 0`, `preferredHeight <= 0`, or `preferredFormat == experimental::io2d::format::invalid`.

`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.

10 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;
```

11 *Effects:* Constructs an object of type `display_surface`.

12 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.

13 The preferredFormat parameter specifies the preferred pixel format value for Draw Format.

14 The value of Scaling Type shall be the value of `scl`.

15 The value of Refresh Rate shall be the value of `rr`.

16 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.

17 All other observable state data shall have their default values.

18 *Throws:* As specified in Error reporting (3).

19 *Error conditions:* `errc::invalid_argument` if `preferredWidth <= 0`, `preferredHeight <= 0`, `preferredDisplayWid <= 0`, `preferredDisplayHeight <= 0`, or `preferredFormat == experimental::io2d::format::invalid`.

`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.

20 Other errors, if any, produced by this function are implementation-defined.

10.13.6 `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 19 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 (3).

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 19 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 (3).

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 19 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 (3).

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 19 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 (3).

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 19 with `w` treated as being the preferred Draw Width and Draw Height is set as specified by Table 19 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 (3).

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 19 with `w` treated as being the preferred Display Height and Display Height is set as specified by Table 19 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 (3).

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(experimental::nullvalue_t) noexcept;
```

54 *Effects:* Sets the Letterbox Brush to its default value.

```
void letterbox_brush(const rgba_color& c);
void letterbox_brush(const rgba_color& c, error_code& ec) noexcept;
```

55 *Effects:* Sets the Letterbox Brush to a value as if `experimental::io2d::brush{ solid_color_brush_factory{ } }`.

56 *Throws:* As specified in Error reporting (3).

57 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void letterbox_brush(const experimental::io2d::brush& b);
void letterbox_brush(const experimental::io2d::brush& b, error_code& ec)
  noexcept;
```

58 *Effects:* Sets the Letterbox Brush to `b`.

59 *Throws:* As specified in Error reporting (3).

60 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void auto_clear(bool val) noexcept;
```

61 *Effects:* Sets Auto Clear to the value of `val`.

```
void refresh_rate(experimental::io2d::refresh_rate rr) noexcept;
```

62 *Effects:* Sets the Refresh Rate to the value of `rr`.

```
bool desired_frame_rate(double fps) noexcept;
```

63 *Effects:* If `!is_finite(fps)`, this function has no effects.

64 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`.

65 *Returns:* `false` if the Desired Frame Rate was set to the value of `fps`; otherwise `true`.

```
void redraw_required() noexcept;
```

66 *Effects:* When `display_surface::show` is executing, informs the implementation that it shall call the Draw Callback as soon as possible.

```
int show();
int show(error_code& ec) noexcept;
```

67 *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.

68 If `display_surface::exit_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.

69 No later than when this function returns, the output device shall cease to display the contents of the Display Buffer.

70 What the output device shall display when it is not displaying the contents of the Display Buffer is unspecified.

71 *Returns:* The possible values and meanings of the possible values returned are implementation-defined.

72 *Throws:* As specified in Error reporting (3).

73 *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.

74 *Error conditions:* `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.

76 Other errors, if any, produced by this function are implementation-defined.

```
void exit_show();
void exit_show(error_code& ec) noexcept;
void exit_show(std::chrono::duration d);
void exit_show(std::chrono::duration d, error_code& ec) noexcept;
```

77 *Requires:* This function shall only be called from the Draw Callback; no diagnostic is required.

78 *Effects:* The implementation shall initiate the process of exiting the `display_surface::show` function's continuous loop.

79 Implementations shall not wait until the `display_surface::show` function's continuous loop ends before returning from this function.

80 Implementations should follow any procedures that the host environment requires in order to cause the `display_surface::show` function's continuous loop to stop executing without error.

81 A *termination time duration* shall then be determined as follows:

(81.1) — If no `std::chrono::duration` is provided, the termination time duration is unspecified. Implementations should exit the `display_surface::show` function's continuous loop as soon as the host environment allows.

(81.2) — Otherwise, implementations shall exit the `display_surface::show` function's continuous loop as soon as the host environment allows once the `std::chrono::duration` value has elapsed.

82 The implementation shall continue to execute the `display_surface::show` function until it returns or until termination time duration milliseconds have passed since the termination time duration was determined, whichever comes first.

83 If the `display_surface::show` function has not returned before termination time duration milliseconds have passed since the termination time duration was determined the implementation shall force the `display_surface::show` function's continuous loop to stop executing and shall then cause `display_surface::show` to return.

10.13.7 `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.

```
tuple<int, int> dimensions() const noexcept;
```

6 *Returns:* A `tuple<int, int>` where the first element is the Draw Width and the second element is the Draw Height.

```
tuple<int, int> display_dimensions() const noexcept;
```

7 *Returns:* A `tuple<int, int>` where the first element is the Display Width and the second element is the Display Height.

```
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 (3).

11 *Error conditions:* `errc::not_enough_memory` if a failure to allocate memory occurs.

```
experimental::io2d::brush letterbox_brush() const noexcept;
```

12 *Returns:* The Letterbox Brush.

```
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`.

10.14 Class `mapped_surface` [mappedsurface]

10.14.1 `mapped_surface` synopsis [mappedsurface.synopsis]

```
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class mapped_surface {
  public:
    // 10.14.3, 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();

    // 10.14.4, 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;

    // 10.14.5, 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;
  };
} } } }
```

10.14.2 `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 (10.7).

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.

10.14.3 `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 *Notes:* It is recommended that users use the `mapped_surface::commit_changes` function to commit changes prior to the destruction of the `mapped_surface` object to ensure consistent behavior.

10.14.4 `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 (3).

3 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
unsigned char* data();
unsigned char* data(error_code& ec) noexcept;
```

4 *Returns:* A native-endian pointer to the pixel data array. [*Example:* Given the following code:

```
image_surface imgsfc{ format::argb32, 100, 100 };
imgsfc.paint(rgba_color::red());
imgsfc.flush();
imgsfc.map([](mapped_surface& mapsfc) -> void {
    auto pixelData = mapsfc.data();
    auto p0 = static_cast<uint32_t>(pixelData[0]);
    auto p1 = static_cast<uint32_t>(pixelData[1]);
    auto p2 = static_cast<uint32_t>(pixelData[2]);
```

```
    auto p3 = static_cast<uint32_t>(pixelData[3]);
    printf("%X %X %X %X\n", p0, p1, p2, p3);
});
```

In a little-endian environment, `p0 == 0x0`, `p1 == 0x0`, `p2 == 0xFF`, and `p3 == 0xFF`.

In a big-endian environment, `p0 == 0xFF`, `p1 == 0xFF`, `p2 == 0x0`, `p3 == 0x0`. — *end example*]

5 *Throws:* As specified in Error reporting (3).

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`.

10.14.5 `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(rgba_color::red());
imgsfc.flush();
imgsfc.map([](mapped_surface& mapsfc) -> void {
    auto pixelData = mapsfc.data();
    auto p0 = static_cast<uint32_t>(pixelData[0]);
    auto p1 = static_cast<uint32_t>(pixelData[1]);
    auto p2 = static_cast<uint32_t>(pixelData[2]);
    auto p3 = static_cast<uint32_t>(pixelData[3]);
    printf("%X %X %X %X\n", p0, p1, p2, p3);
});
```

In a little-endian environment, `p0 == 0x0`, `p1 == 0x0`, `p2 == 0xFF`, and `p3 == 0xFF`.

In a big-endian environment, `p0 == 0xFF`, `p1 == 0xFF`, `p2 == 0x0`, `p3 == 0x0`. — *end example*]

2 *Throws:* As specified in Error reporting (3).

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`.

11 Standalone functions [io2d.standalone]

11.1 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
```

11.2 `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`.

11.3 `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 (3).

3 *Error conditions:* The errors, if any, produced by this function are the same as the errors for the equivalent `display_surface` constructor (10.13.5).

11.4 `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 (3).

3 *Error conditions:* The errors, if any, produced by this function are the same as the errors for the equivalent `display_surface` constructor (10.12.3).

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

Index of library names

Index of implementation-defined behavior

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