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

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

**Contents**

List of Tables

List of Figures

1 General
   1.1 Scope ......................................................... 1
   1.2 Normative references ........................................ 1
   1.3 Terms and definitions ........................................ 1

2 Requirements
   2.1 Namespaces and headers ...................................... 7
   2.2 Feature test macros .......................................... 7
   2.3 Native handles ................................................ 7
   2.4 IEC 559 floating point support .............................. 7
   2.5 Exact width integer types ................................... 7

3 Error reporting .................................................. 9

4 Header `<experimental/io2d>` synopsis .......................... 11

5 Error codes
   5.1 Enum class `io2d_error` ........................................ 14
   5.2 Class `io2d_error_category` .................................. 16

6 Colors
   6.1 Class `rgba_color` ............................................. 18
   6.2 `literals` namespace ........................................... 36

7 Class `vector_2d`
   7.1 `vector_2d` synopsis .......................................... 37
   7.2 `vector_2d` Description ....................................... 37
   7.3 `vector_2d` constructors and assignment operators ........ 37
   7.4 `vector_2d` modifiers ......................................... 38
   7.5 `vector_2d` observers .......................................... 38
   7.6 `vector_2d` member operators ................................ 38
   7.7 `vector_2d` non-member operators ............................ 39

8 Class `rectangle`
   8.1 `rectangle` synopsis ......................................... 40
   8.2 `rectangle` Description ...................................... 40
   8.3 `rectangle` constructors and assignment operators ....... 40
   8.4 `rectangle` modifiers ......................................... 41
   8.5 `rectangle` observers ......................................... 42

9 Class `matrix_2d` ................................................ 43
List of Tables

1  io2d_error enumerator meanings .................................................. 14
2  native_geometry_collection state data ........................................ 49
3  path_data_type enumerator meanings ......................................... 55
4  font_slant enumerator meanings .................................................. 90
5  font_weight enumerator meanings .............................................. 91
6  subpixel_order enumerator meanings .......................................... 92
7  extend enumerator meanings ....................................................... 107
8  filter enumerator meanings ....................................................... 108
9  brush_type enumerator meanings ............................................... 108
10 antialias enumerator meanings ................................................... 120
11 content value meanings ............................................................. 121
12 fill_rule enumerator meanings ................................................... 122
13 line_cap enumerator meanings ................................................... 122
14 line_join enumerator meanings ................................................. 123
15 compositing_operator basic enumerator meanings ............................ 125
16 compositing_operator blend enumerator meanings ........................... 126
17 compositing_operator hsl enumerator meanings ............................... 129
18 format enumerator meanings ....................................................... 130
19 scaling enumerator meanings ...................................................... 132
20 Surface observable state ............................................................. 144
21 surface rendering and composing operations ................................... 146
22 Display surface observable state ............................................... 172
List of Figures
1 General

1.1 Scope

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

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.

1.3 Terms and definitions

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

1.3.1 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 visual data

data representing color, transparency, or some combination thereof

1.3.3 channel

a bounded set of homogeneously-spaced real numbers in the range [0, 1]
1.3.4 visual data format
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 alpha
alpha
visual data representing transparency

1.3.6 pixel
pixel
a discrete visual data element with a visual data format-dependent composition

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

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

1.3.9 anti-aliasing
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]

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

1.3.11 closed path geometry
closed path geometry
a path geometry with one or more path segments where the initial path segment’s start point is used to define the end point of the final path segment

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

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

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

§ 1.3.14
1.3.15 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 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 sRGB color space

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

1.3.18 composing operation

an operation that uses a composition algorithm to combine part or all of a source of visual data capable of being treated as though it were a pixmap with a pixmap

1.3.19 composition algorithm

an algorithm that combines a source pixel and a destination pixel producing a result that has the same visual data format as the destination pixel

1.3.20 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

1.3.21 current point

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

1.3.22 degenerate path geometry

a path geometry with only one path segment [Note: The path segment is not required to be a degenerate path segment. —end note]
1.3.23  degenerate path segment
a path segment which has the same value for its start point and its end point is a degenerate path segment

1.3.24  filter
a mathematical function that determines the pixel value of a point for a pixmap

1.3.25  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]

1.3.26  graphics data
<graphics data> visual data stored in an unspecified form

1.3.27  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)

1.3.28  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]

1.3.29  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

1.3.30  graphics resource
<path geometry graphics resource> an object of unspecified type used by an implementation to store a collection of zero or more path geometries in an unspecified format

1.3.31  pixmap
a raster graphics data graphics resource

1.3.32  point
<point> a coordinate designated by a floating point x axis value and a floating point y axis value within the standard coordinate space

1.3.33  point
<integral point> a coordinate designated by an integral x axis value and an integral y axis value within the standard coordinate space
1.3.34  [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]

1.3.35  [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

1.3.36  [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]

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

1.3.38  [io2d.general.defns.lastmovetopoint]  
last-move-to point
the point in a path geometry that is the start point of the initial path segment

1.3.39  [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]

1.3.40  [io2d.general.defns.openpathgeometry]  
open path geometry
a path geometry with one or more path segments where the initial path segment’s start point is not used to define the end point of the 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 geometry is still an open path geometry 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]

1.3.41  [io2d.general.defns.pathgeometry]  
path geometry
a collection of path segments where the end point of each path segment, except the final path segment, shall be used to define the start point of exactly one other path segment in the collection

1.3.42  [io2d.general.defns.pathsegment]  
path segment
is a line or a curve, each of which has a start point and an end point
1.3.43 \[io2d.general.defns.render\]
render
to transform path geometries or text into raster graphics data in the manner specified by a set of graphics state data

1.3.44 \[io2d.general.defns.renderingoperation\]
rendering operation
an operation that performs rendering

1.3.45 \[io2d.general.defns.sample\]
sample
to use a filter to obtain a pixel for a given coordinate from a pixmap

1.3.46 \[io2d.general.defns.colorstop\]
color stop
a tuple composed of a floating point offset value in the range \([0, 1]\) and a color value
2 Requirements

2.1 Namespaces and headers

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

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

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

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

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

```
#define __cpp_lib_experimental_io2d 201507
```

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

2.3 Native handles

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

In order to implement this Technical Specification, `numeric_limits<double>::is_iec559 == true` shall evaluate to `true`.

2.5 Exact width integer types

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

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.

[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]

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.

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 {
  // From C++ Extensions for Library Fundamentals, N4335
  struct nullopt_t;
  constexpr nullopt_t nullopt{ implementation-defined }; 

  namespace io2d { inline namespace v1 {

    typedef tuple<vector<double>, double> dashes;

    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 extend;
    enum class filter;
    enum class brush_type;
    enum class font_slant;
    enum class font_weight;
    enum class subpixel_order;
    enum class path_data_type;
    enum class scaling;

    class io2d_error_category;
    const error_category& io2d_category() noexcept;

    class rectangle;

    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;
  }
}

Header `<experimental/io2d> synopsis
class font_extents;
class text_extents;

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);

class path_data_item;
class path_data_item::arc;
class path_data_item::arc_negative;
class path_data_item::change_matrix;
class path_data_item::change_origin;
class path_data_item::close_path;
class path_data_item::curve_to;
class path_data_item::line_to;
class path_data_item::move_to;
class path_data_item::new_sub_path;
class path_data_item::path_data;
class path_data_item::rel_curve_to;
class path_data_item::rel_line_to;
class path_data_item::rel_move_to;

class path;
class path_factory;

class device;

class font_options;

class font_face;
class simple_font_face;

class brush;
class solid_color_brush_factory;
class linear_brush_factory;
class radial_brush_factory;
class surface_brush_factory;

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

int format_stride_for_width(format format, int width) noexcept;
display_surface make_display_surface(int preferredWidth,
int preferredHeight, format preferredFormat,
scaling sc1 = scaling::letterbox);
display_surface make_display_surface(int preferredWidth,
int preferredHeight, format preferredFormat, error_code& ec,
scaling sc1 = scaling::letterbox) noexcept;
display_surface make_display_surface(int preferredWidth,
int preferredHeight, format preferredFormat, int preferredDisplayWidth,
```cpp
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

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]

```cpp
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    enum class io2d_error {
        success,
        invalid_restore,
        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,
        user_font_immutable,
        user_font_error,
        invalid_clusters,
        device_error,
        invalid_mesh_construction,
    };
}}}
```

```cpp
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

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>The operation completed successfully.</td>
</tr>
</tbody>
</table>

§ 5.1.3
<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid_restore</td>
<td>A call was made to <code>surface::restore</code> for which no prior call to <code>surface::save</code> was made.</td>
</tr>
<tr>
<td>no_current_point</td>
<td>The operation requires a current point but no current point was set. This is the result of an improper call to <code>path_factory::rel_curve_to</code>, <code>path_factory::rel_line_to</code>, or <code>path_factory::rel_move_to</code> or is the result of a call to <code>path_factory::append</code> with malformed data. See</td>
</tr>
<tr>
<td>invalid_matrix</td>
<td>A <code>matrix_2d</code> value that the operation depends on is invalid. Except as otherwise specified, this means that the <code>matrix_2d</code> value is not invertible.</td>
</tr>
<tr>
<td>invalid_status</td>
<td>An internal error has occurred. The conditions and circumstances under which this <code>io2d_error</code> value occurs are implementation-defined. [Note: This value should only be used when no other <code>io2d_error</code> value is appropriate. It signifies that an implementation-specific error occurred such as passing a bad native handle as an argument. — end note]</td>
</tr>
<tr>
<td>null_pointer</td>
<td>A null pointer value was unexpectedly encountered. The conditions and circumstances under which this <code>io2d_error</code> value occurs are implementation-defined.</td>
</tr>
<tr>
<td>invalid_string</td>
<td>A UTF-8 string value was expected but the string is not a valid UTF-8 string.</td>
</tr>
<tr>
<td>invalid_path_data</td>
<td>Invalid data was encountered in a <code>path</code> or a <code>path_factory</code> object. [Note: This status value should only occur when a user creates invalid path data and appends it to a path. — end note]</td>
</tr>
<tr>
<td>read_error</td>
<td>An error occurred while attempting to read data from an input stream.</td>
</tr>
<tr>
<td>write_error</td>
<td>An error occurred while attempting to write data to an output stream.</td>
</tr>
<tr>
<td>surface_finished</td>
<td>An attempt was made to use or manipulate a <code>surface</code> object or <code>surface</code>-derived object which is no longer valid. [Note: This can occur due to a previous call to <code>surface::finish</code> or as a result of erroneous usage of a native handle. — end note]</td>
</tr>
<tr>
<td>invalid_dash</td>
<td>An invalid dash value was specified in a call to <code>surface::set_dashes</code>.</td>
</tr>
<tr>
<td>invalid_index</td>
<td>An index value was specified in a call to a function which is outside the range of index values that are currently valid.</td>
</tr>
<tr>
<td>clip_not_representable</td>
<td>A call was made to <code>surface::get_clip_rectangles</code> when the <code>surface</code> object’s current clipping region could not be represented with rectangles.</td>
</tr>
<tr>
<td>invalid_stride</td>
<td>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 <code>format_stride_for_width</code> to obtain the correct stride value.</td>
</tr>
</tbody>
</table>

§ 5.1.3
Table 1 — io2d_error enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_font_immutable</td>
<td>User font immutable. [Note: Reserved. — end note]</td>
</tr>
<tr>
<td>user_font_error</td>
<td>User font error. [Note: Reserved. — end note]</td>
</tr>
<tr>
<td>invalid_clusters</td>
<td>A call was made to surface::show_text_glyphs with a std::vector&lt;text_clusters&gt; argument that does not properly map the UTF-8 std::string code points to the std::vector&lt;glyph&gt; glyphs.</td>
</tr>
<tr>
<td>device_error</td>
<td>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]</td>
</tr>
</tbody>
</table>

5.2 Class io2d_error_category

5.2.1 io2d_error_category 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

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

virtual const char* name() const noexcept override;

Returns: A pointer to the string "io2d".

virtual string message(int errVal) const override;

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

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

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]

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

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

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

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.

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".

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

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

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

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

§ 5.2.4
6 Colors

1 This section is forthcoming in a future revision.

6.1 Class rgba_color

1 The class \texttt{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 \texttt{double}.
3 There is also an alpha channel, which is a \texttt{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

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class rgba\_color {
    // 6.1.2, construct/copy/move/destroy:
    rgba\_color() noexcept;
    rgba\_color(double r, double g, double b, double a = 1.0);
    rgba\_color(double r, double g, double b, error\_code\& ec) noexcept;
    rgba\_color(double r, double g, double b, double a, error\_code\& ec) noexcept;
    rgba\_color(const rgba\_color\& c) noexcept;
    rgba\_color\& operator=(const rgba\_color\& c) noexcept;
    rgba\_color\& operator=(rgba\_color\& c) noexcept;
    // 6.1.3, modifiers:
    void r(double val);
    void r(double val, error\_code\& ec) noexcept;
    void g(double val);
    void g(double val, error\_code\& ec) noexcept;
    void b(double val);
    void b(double val, error\_code\& ec) noexcept;
    void a(double val);
    void a(double val, error\_code\& ec) 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& 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;

§ 6.1.1
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;

private:
    double _Red;    // exposition only
    double _Green;  // exposition only
    double _Blue;   // exposition only
    double _Alpha;  // exposition only
};

// 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() noexcept;

Effects: Constructs an object of type rgba_color.

Postconditions: _Red == 0.0.
    _Green == 0.0.
    _Blue == 0.0.
    _Alpha == 1.0.

Note: The resulting color is opaque black, which can also be obtained from rgba_color::black().

§ 6.1.2 21
rgba_color(double r, double g, double b, double a = 1.0);
rgba_color(double r, double g, double b, error_code& ec) noexcept;
rgba_color(double r, double g, double b, double a, error_code& ec) noexcept;

Requires: \( r \geq 0.0 \) and \( r \leq 1.0 \) and \( g \geq 0.0 \) and \( g \leq 1.0 \) and \( b \geq 0.0 \) and \( b \leq 1.0 \). Where there is an \( a \) parameter, \( a \geq 0.0 \) and \( a \leq 1.0 \).

Effects: Constructs an object of type rgba_color.

Postconditions: Where there is an \( a \) parameter, \_Alpha = a; otherwise \_Alpha = 1.0.

\_Red = r * a.
\_Green = g * a.
\_Blue = b * a.

Throws: As specified in Error reporting (3).

Remarks: In the event of a non-throwing error, the object shall be constructed to meet the following conditions:

\_Red == 1.0 && \_Green == 0.0 && \_Blue == 1.0 && \_Alpha == 1.0.

Error conditions: errc::argument_out_of_domain if the value of \( r, g, b, \) or \( a \) fails to meet the preconditions.

Notes: When an object is constructed despite the presence of one or more erroneous arguments, the resulting color is opaque magenta, which can also be obtained from rgba_color::magenta().

This color was picked because it is a legal color value, it does not have all channels set to the same value (which would hide, e.g., erroneous saturation calculations), and in the hope that it will help highlight error visibility (both visually and in an examination of the data) in the event that the developer neglects to check the error_code& argument following construction of the object.

### 6.1.3 rgba_color modifiers

void r(double val);
void r(double val, error_code& ec) noexcept;

Requires: \( val \geq 0.0 \) and \( val \leq 1.0 \).

Throws: As specified in Error reporting (3).

Postconditions: \_Red == val.

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::argument_out_of_domain if the value of \( val \) fails to meet the preconditions.

void g(double val);
void g(double val, error_code& ec) noexcept;

Requires: \( val \geq 0.0 \) and \( val \leq 1.0 \).

Throws: As specified in Error reporting (3).

Postconditions: \_Green == val.

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::argument_out_of_domain if the value of \( val \) fails to meet the preconditions.

void b(double val);
void b(double val, error_code& ec) noexcept;
Requires: val >= 0.0 and val <= 1.0.

Throws: As specified in Error reporting (3).

Postconditions: _Blue == val.

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::argument_out_of_domain if the value of val fails to meet the preconditions.

void a(double val);
void a(double val, error_code& ec) noexcept;

Requires: val >= 0.0 and val <= 1.0.

Throws: As specified in Error reporting (3).

Postconditions: _Alpha == val.
_Red = _Red * val.
_Green = _Green * val.
_Blue = _Blue * val.

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::argument_out_of_domain if the value of val fails to meet the preconditions.

6.1.4 rgba_color observers

 double r() const noexcept;
   Returns: _Red.

double g() const noexcept;
   Returns: _Green.

double b() const noexcept;
   Returns: _Blue.

double a() const noexcept;
   Returns: _Alpha.

6.1.5 rgba_color static member functions

 static const rgba_color& alice_blue() noexcept;
   Returns: a const reference to the static rgba_color object rgba_color{ 240ubyte, 248ubyte, 255ubyte, 255ubyte }.

 static const rgba_color& antique_white() noexcept;
   Returns: a const reference to the static rgba_color object rgba_color{ 250ubyte, 235ubyte, 215ubyte, 255ubyte }.

 static const rgba_color& aqua() noexcept;
   Returns: a const reference to the static rgba_color object rgba_color{ 0ubyte, 255ubyte, 255ubyte, 255ubyte }.

 static const rgba_color& aquamarine() noexcept;
Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 127ubyte, 255ubyte, 212ubyte, 255ubyte }.

static const RGBA_COLOR& azure() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 240ubyte, 255ubyte, 255ubyte, 255ubyte }.

static const RGBA_COLOR& beige() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 245ubyte, 245ubyte, 220ubyte, 255ubyte }.

static const RGBA_COLOR& bisque() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 255ubyte, 228ubyte, 196ubyte, 255ubyte }.

static const RGBA_COLOR& black() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 0ubyte, 0ubyte, 0ubyte, 255ubyte }.

static const RGBA_COLOR& blanched_almond() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 255ubyte, 235ubyte, 205ubyte, 255ubyte }.

static const RGBA_COLOR& blue() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 0ubyte, 0ubyte, 255ubyte, 255ubyte }.

static const RGBA_COLOR& blue_violet() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 138ubyte, 43ubyte, 226ubyte, 255ubyte }.

static const RGBA_COLOR& brown() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 165ubyte, 42ubyte, 42ubyte, 255ubyte }.

static const RGBA_COLOR& burly_wood() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 222ubyte, 184ubyte, 135ubyte, 255ubyte }.

static const RGBA_COLOR& cadet_blue() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 95ubyte, 158ubyte, 160ubyte, 255ubyte }.

static const RGBA_COLOR& chartreuse() noexcept;

Returns: a const reference to the static_RGBA_COLOR object _RGBA_COLOR{ 127ubyte, 255ubyte, 0ubyte, 255ubyte }.

static const RGBA_COLOR& chocolate() noexcept;
Returns: a const reference to the static `rgba_color` object `rgba_color{ 210ubyte, 105ubyte, 30ubyte, 255ubyte }`.

static const `rgba_color` & coral() noexcept;

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

static const `rgba_color` & cornflower_blue() noexcept;

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

static const `rgba_color` & cornsilk() noexcept;

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

static const `rgba_color` & crimson() noexcept;

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

static const `rgba_color` & cyan() noexcept;

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

static const `rgba_color` & dark_blue() noexcept;

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

static const `rgba_color` & dark_cyan() noexcept;

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

static const `rgba_color` & dark_goldenrod() noexcept;

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

static const `rgba_color` & dark_gray() noexcept;

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

static const `rgba_color` & dark_green() noexcept;

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

static const `rgba_color` & dark_grey() noexcept;

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

static const `rgba_color` & dark_khaki() noexcept;
Returns: a const reference to the static `rgba_color` object `rgba_color{ 189ubyte, 183ubyte, 107ubyte, 255ubyte }`.

static const `rgba_color`& dark_magenta() noexcept;

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

static const `rgba_color`& dark_olive_green() noexcept;

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

static const `rgba_color`& dark_orange() noexcept;

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

static const `rgba_color`& dark_orchid() noexcept;

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

static const `rgba_color`& dark_red() noexcept;

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

static const `rgba_color`& dark_salmon() noexcept;

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

static const `rgba_color`& dark_sea_green() noexcept;

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

static const `rgba_color`& dark_slate_blue() noexcept;

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

static const `rgba_color`& dark_slate_gray() noexcept;

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

static const `rgba_color`& dark_slate_grey() noexcept;

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

static const `rgba_color`& dark_turquoise() noexcept;

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

static const `rgba_color`& dark_violet() noexcept;
Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{148ubyte, 0ubyte, 211ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{deep\_pink()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{255ubyte, 20ubyte, 147ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{deep\_sky\_blue()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{0ubyte, 191ubyte, 255ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{dim\_gray()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{105ubyte, 105ubyte, 105ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{dim\_grey()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{105ubyte, 105ubyte, 105ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{dodger\_blue()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{30ubyte, 144ubyte, 255ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{firebrick()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{178ubyte, 34ubyte, 34ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{floral\_white()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{255ubyte, 250ubyte, 240ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{forest\_green()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{34ubyte, 139ubyte, 34ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{fuchsia()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{255ubyte, 0ubyte, 255ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{gainsboro()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{220ubyte, 220ubyte, 220ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{ghost\_white()} noexcept;

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{248ubyte, 248ubyte, 255ubyte, 255ubyte\}}.

static const \texttt{rgba\_color}& \texttt{gold()} noexcept;

\textsection{6.1.5}
Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 215ubyte, 0ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& goldenrod() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 218ubyte, 165ubyte, 32ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& gray() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 128ubyte, 128ubyte, 128ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& green() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 0ubyte, 128ubyte, 0ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& green\_yellow() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 173ubyte, 255ubyte, 47ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& grey() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 128ubyte, 128ubyte, 128ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& honeydew() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 240ubyte, 255ubyte, 240ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& hot\_pink() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 105ubyte, 180ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& indian\_red() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 205ubyte, 92ubyte, 92ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& indigo() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 75ubyte, 0ubyte, 130ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& ivory() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 255ubyte, 240ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& khaki() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 240ubyte, 230ubyte, 140ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color& lavender() noexcept;
\end{verbatim}
Returns: a const reference to the static `rgba_color` object `rgba_color{ 230ubyte, 230ubyte, 250ubyte, 255ubyte }`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```cpp
static const rgba_color& light_salmon() noexcept;
```
Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 255ubyte, 160ubyte, 122ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& light\_sea\_green()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 32ubyte, 178ubyte, 170ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& light\_sky\_blue()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 135ubyte, 206ubyte, 250ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& light\_slate\_gray()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 119ubyte, 136ubyte, 153ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& light\_slate\_grey()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 119ubyte, 136ubyte, 153ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& light\_steel\_blue()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 176ubyte, 196ubyte, 222ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& light\_yellow()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 255ubyte, 255ubyte, 224ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& lime()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 0ubyte, 255ubyte, 0ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& lime\_green()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 50ubyte, 205ubyte, 50ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& linen()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 250ubyte, 240ubyte, 230ubyte, 255ubyte \}}.

\texttt{static const rgba\_color&()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 255ubyte, 0ubyte, 255ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& magenta()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 255ubyte, 0ubyte, 255ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& maroon()} noexcept;

Returns: a const reference to the static rgba_color object \texttt{rgba\_color\{ 128ubyte, 0ubyte, 0ubyte, 255ubyte \}}.

\texttt{static const rgba\_color& medium\_aquamarine()} noexcept;

§ 6.1.5
static const rgba_color& medium_blue() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 102ubyte, 205ubyte, 170ubyte, 255ubyte }.

static const rgba_color& medium_orchid() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 0ubyte, 0ubyte, 205ubyte, 255ubyte }.

static const rgba_color& medium_purple() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 186ubyte, 85ubyte, 211ubyte, 255ubyte }.

static const rgba_color& medium_sea_green() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 60ubyte, 179ubyte, 113ubyte, 255ubyte }.

static const rgba_color& medium_slate_blue() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 123ubyte, 104ubyte, 238ubyte, 255ubyte }.

static const rgba_color& medium_spring_green() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 0ubyte, 250ubyte, 154ubyte, 255ubyte }.

static const rgba_color& medium_turquoise() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 72ubyte, 209ubyte, 204ubyte, 255ubyte }.

static const rgba_color& medium_violet_red() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 199ubyte, 21ubyte, 133ubyte, 255ubyte }.

static const rgba_color& midnight_blue() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 25ubyte, 25ubyte, 112ubyte, 255ubyte }.

static const rgba_color& mint_cream() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 245ubyte, 255ubyte, 250ubyte, 255ubyte }.

static const rgba_color& misty_rose() noexcept;
Returns: a const reference to the static rgba_color object rgba_color{ 255ubyte, 228ubyte, 225ubyte, 255ubyte }.
Returns: a const reference to the static `rgba_color` object `rgba_color{ 255ubyte, 228ubyte, 181ubyte, 255ubyte }`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```cpp
static const rgba_color& pale_violet_red() noexcept;
```
Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 219ubyte, 112ubyte, 147ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& papaya\_whip() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 239ubyte, 213ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& peach\_puff() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 218ubyte, 185ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& peru() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 205ubyte, 133ubyte, 63ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& pink() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 192ubyte, 203ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& plum() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 221ubyte, 160ubyte, 221ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& powder\_blue() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 176ubyte, 224ubyte, 230ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& purple() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 128ubyte, 0ubyte, 128ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& red() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 255ubyte, 0ubyte, 0ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& rosy\_brown() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 188ubyte, 143ubyte, 143ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& royal\_blue() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 65ubyte, 105ubyte, 225ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& saddle\_brown() noexcept;
\end{verbatim}

Returns: a const reference to the static \texttt{rgba\_color} object \texttt{rgba\_color\{ 139ubyte, 69ubyte, 19ubyte, 255ubyte \}}.

\begin{verbatim}
static const rgba\_color\& salmon() noexcept;
\end{verbatim}

§ 6.1.5
Returns: a const reference to the static `rgba_color` object `rgba_color{ 250ubyte, 128ubyte, 114ubyte, 255ubyte }`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

§ 6.1.5
Returns: a const reference to the static `rgba_color` object `rgba_color{ 70ubyte, 130ubyte, 180ubyte, 255ubyte }`.

static const `rgba_color`& tan() noexcept;

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

static const `rgba_color`& teal() noexcept;

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

static const `rgba_color`& thistle() noexcept;

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

static const `rgba_color`& tomato() noexcept;

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

static const `rgba_color`& transparent_black() noexcept;

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

static const `rgba_color`& turquoise() noexcept;

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

static const `rgba_color`& violet() noexcept;

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

static const `rgba_color`& wheat() noexcept;

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

static const `rgba_color`& white() noexcept;

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

static const `rgba_color`& white_smoke() noexcept;

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

static const `rgba_color`& yellow() noexcept;

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

static const `rgba_color`& yellow_green() noexcept;

Returns: a const reference to the static `rgba_color` object `rgba_color{ 154ubyte, 205ubyte, 50ubyte, 255ubyte }`.
6.1.6  rgba\_color non-member operators

bool operator\==(const rgba\_color& lhs, const rgba\_color& rhs) noexcept;

1 \textit{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 \textit{Returns:} !(lhs == rhs)

6.2  literals namespace

6.2.1  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

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

1 \textit{Returns:} max(0.0, min(1.0, static\_cast<double>(value) / 255.0))

double operator ") unorm(long double value);

2 \textit{Returns:} nearbyint(max(0.0, min(1.0, static\_cast<double>(value))) * 255.0) / 255.0
7 Class vector_2d

7.1 vector_2d synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class vector_2d {
  public:
    // 7.3, construct/copy/move/destroy:
    vector_2d() noexcept;
    vector_2d(double x, double y) noexcept;
    vector_2d(const vector_2d&) noexcept;
    vector_2d& operator=(const vector_2d&) noexcept;
    vector_2d(vector_2d&&) noexcept;
    vector_2d& operator=(vector_2d&&) noexcept;

    // 7.4, modifiers:
    void x(double value) noexcept;
    void y(double value) noexcept;

    // 7.5, 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;

    // 7.6, member operators:
    vector_2d& operator+=(const vector_2d& rhs) noexcept;
    vector_2d& operator-=(const vector_2d& rhs) noexcept;
    vector_2d& operator*=(double rhs) noexcept;
  private:
    double _X; // exposition only
    double _Y; // exposition only
  };

  // 7.7, 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, 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, double rhs) noexcept;
  vector_2d operator*(double lhs, const vector_2d& rhs) noexcept;

} } } }

7.2 vector_2d Description

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

7.3 vector_2d constructors and assignment operators

§ 7.3
vector_2d() noexcept;

   Effects: Constructs an object of type vector_2d.
   Postconditions: _X == 0.0 && _Y == 0.0.

vector_2d(double x, double y) noexcept;

   Effects: Constructs an object of type vector_2d.
   Postconditions: _X == x && _Y == y.

7.4 vector_2d modifiers

void x(double value) noexcept;

   Postconditions: _X == value.

void y(double value) noexcept;

   Postconditions: _Y == value.

7.5 vector_2d observers

double x() const noexcept;

   Returns: _X.

double y() const noexcept;

   Returns: _Y.

double length() const noexcept;

   Returns: sqrt(_X * _X + _Y * _Y).

double dot(const vector_2d& other) const noexcept;

   Returns: _X * other._X + _Y * other._Y.

vector_2d to_unit() const noexcept;

   Returns: vector_2d{ _X / length(), _Y / length() }.

7.6 vector_2d member operators

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

   Effects: *this = *this + rhs.
   Returns: *this.

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

   Effects: *this = *this - rhs.
   Returns: *this.

vector_2d& operator*=(double rhs) noexcept;

   Effects: *this = *this * rhs.
   Returns: *this.
### 7.7 vector_2d non-member operators

[vector2d.ops]

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

    Returns: lhs.x() == rhs.x() && lhs.y() == rhs.y().

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

    Returns: !(lhs == rhs).

    vector_2d operator+(const vector_2d& lhs) noexcept;

    Returns: vector_2d(lhs).

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

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

    vector_2d operator-(const vector_2d& lhs) noexcept;

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

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

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

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

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

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

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

8.1 rectangle synopsis [rectangle.synopsis]

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class rectangle {
    public:
        // 8.3, construct/copy/move/destroy:
        rectangle() noexcept;
        rectangle(double x, double y, double width, double height) noexcept;
        rectangle(const vector_2d& t1, const vector_2d& br) noexcept;
        rectangle(const rectangle& r) noexcept;
        rectangle& operator=(const rectangle& r) noexcept;
        rectangle(rectangle&& r) noexcept;
        rectangle& operator=(rectangle&& r) noexcept;
    
        // 8.4, modifiers:
        void x(double value) noexcept;
        void y(double value) noexcept;
        void width(double value) noexcept;
        void height(double value) noexcept;
        void top_left(const vector_2d& value) noexcept;
        void bottom_right(const vector_2d& value) noexcept;
    
        // 8.5, 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;
    
    private:
        double _X; // exposition only
        double _Y; // exposition only
        double _Width; // exposition only
        double _Height; // exposition only
    };
} } } }

8.2 rectangle Description [rectangle.intro]
1 The class rectangle describes an object that represents a rectangle.

8.3 rectangle constructors and assignment operators [rectangle.cons]

rectangle() noexcept;
1  Effects: Constructs an object of type rectangle.
2  Postconditions: _X == 0.0.
3       _Y == 0.0.
4       _Width == 0.0.
rectangle(double x, double y, double w, double h) noexcept;

    Effects: Constructs an object of type rectangle.
    Postconditions: _X == x.
    _Y == y.
    _Width == w.
    _Height == h.

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

    Effects: Constructs an object of type rectangle from a top-left coordinate and a bottom-right coordinate.
    Postconditions: _X == tl.x().
    _Y == tl.y().
    _Width == max(0.0, br.x() - tl.x()).
    _Height == max(0.0, br.y() - tl.y()).

8.4 rectangle modifiers

    [rectangle.modifiers]

void x(double value) noexcept;

    Postconditions: _X == value.

void y(double value) noexcept;

    Postconditions: _Y == value.

void width(double value) noexcept;

    Postconditions: _Width == value.

void height(double value) noexcept;

    Postconditions: _Height == value.

void top_left(const vector_2d& value) noexcept;

    Postconditions: _X == value.x().
    _Y == value.y().

void bottom_right(const vector_2d& value) noexcept;

    Postconditions: _Width == max(0.0, value.x() - _X).
    _Height == max(0.0, value.y() - _Y).
8.5 rectangle observers

```cpp
double x() const noexcept;
   // Returns: _X.
double y() const noexcept;
   // Returns: _Y.
double width() const noexcept;
   // Returns: _Width.
double height() const noexcept;
   // Returns: _Height.

vector_2d top_left() const noexcept;
   // Returns: vector_2d{ _X, _Y }.

vector_2d bottom_right() const noexcept;
   // Returns: vector_2d{ _X + _Width, _Y + _Height }.
```

§ 8.5
9  Class matrix_2d

9.1  matrix_2d synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class matrix_2d {
        public:

        // 9.3, construct/copy/move/destroy:
        matrix_2d() noexcept;
        matrix_2d(const matrix_2d& other) noexcept;
        matrix_2d& operator=(const matrix_2d& other) noexcept;
        matrix_2d(matrix_2d&& other) noexcept;
        matrix_2d& operator=(matrix_2d&& other) noexcept;
        matrix_2d(double v00, double v01, double v10, double v11,
                double v20, double v21) noexcept;

        // 9.4, 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;
        static matrix_2d init_shear_y(double factor) noexcept;

        // 9.5, 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;

        // 9.6, 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;
    }
}
}
}
// 9.7, matrix_2d member operators:
matrix_2d& operator*=(const matrix_2d& rhs) noexcept;

private:
    double _M00; // exposition only
    double _M01; // exposition only
    double _M10; // exposition only
    double _M11; // exposition only
    double _M20; // exposition only
    double _M21; // exposition only
};

// 9.8, 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;
}

9.2 matrix_2d Description

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

Mathematically, regardless of the operations performed on a matrix_2d, the third column will always have the column vector value of [0, 0, 0, 1]. As such, it is not included in the observable data of the matrix.

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.

[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.

The layout of the resulting matrix would be as such:
[[ m00 m01 m02 ]]
[[ m10 m11 m12 ]]
[[ m20 m21 m22 ]] — end note]

9.3 matrix_2d constructors and assignment operators

matrix_2d() noexcept;

Effects: Constructs an object of type matrix_2d.

Postconditions: _M00 == 1.0.
_M01 == 0.0.
_M10 == 0.0.
_M11 == 1.0.
_M20 == 0.0.
_M21 == 0.0.

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

§ 9.3
matrix_2d(double v00, double v01, double v10, double v11,
        double v20, double v21) noexcept;

    Effects: Constructs an object of type matrix_2d.
    Postconditions: _M00 == v00.
    _M01 == v01.
    _M10 == v10.
    _M11 == v11.
    _M20 == v20.
    _M21 == v21.

9.4 matrix_2d static factory functions

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).

9.5 matrix_2d modifiers

    void m00(double value) noexcept;
    Postconditions: _M00 == value.

    void m01(double value) noexcept;
    Postconditions: _M01 == value.

    void m10(double value) noexcept;
    Postconditions: _M10 == value.

    void m11(double value) noexcept;
    Postconditions: _M11 == value.

    void m20(double value) noexcept;
    Postconditions: _M20 == value.

    void m21(double value) noexcept;
    Postconditions: _M21 == value.
void m21(double value) noexcept;

Postconditions: _M21 == value.

matrix_2d& translate(const vector_2d& value) noexcept;

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

matrix_2d& scale(const vector_2d& value) 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();

Effects:
const auto det = _M00 * _M11 - _M01 * _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 = _M11 * cM22 - cM12 * _M21;
const auto adjugateM01 = -(cM01 * cM22 - cM12 * cM21);
const auto adjugateM10 = -(cM10 * cM22 - cM12 * cM20);
const auto adjugateM11 = _M00 * cM22 - cM02 * cM21;
const auto adjugateM20 = _M10 * _M21 - _M11 * _M20;
const auto adjugateM21 = -(cM00 * _M21 - cM01 * _M20);

_M00 = inverseDet * adjugateM00;
_M01 = inverseDet * adjugateM01;
_M10 = inverseDet * adjugateM10;
_M11 = inverseDet * adjugateM11;
_M20 = inverseDet * adjugateM20;
_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.
9.6 matrix_2d observers

    double m00() const noexcept;
   ```
   Returns: _M00.
   ```
    double m01() const noexcept;
   ```
   Returns: _M01.
   ```
    double m10() const noexcept;
   ```
   Returns: _M10.
   ```
    double m11() const noexcept;
   ```
   Returns: _M11.
   ```
    double m20() const noexcept;
   ```
   Returns: _M20.
   ```
    double m21() const noexcept;
   ```
   Returns: _M21.
   ```

    bool is_invertible const noexcept;
   ```
   Returns: true if all of the following are true:
   ```
    — isfinite(_M00)
    — isfinite(_M01)
    — isfinite(_M10)
    — isfinite(_M11)
    — isfinite(_M20)
    — isfinite(_M21)
    — (_M00 * _M11 - _M01 * M10) != 0.0
   ```
   Otherwise returns false.

    double determinant() const;
    double determinant(error_code& ec) const noexcept;
   ```
   Returns: _M00 * _M11 - _M01 * M10.
   ```
   In the event of a non-throwing error, the function shall return numeric_limits<double>::quiet_-
   NaN().
   ```
   Throws: As specified in Error reporting (3).
   ```
   Error conditions: io2d_error::invalid_matrix if !isfinite(_M00) || !isfinite(_M01) || !isfinite(_-
   M10) || !isfinite(_M11) || !isfinite(_M20) || !isfinite(_M21).
   ```

    vector_2d transform_distance(const vector_2d& dist) const noexcept;
   ```
   Returns: vector_2d(m00() * dist.x() + m10() * dist.y(), m01() * dist.x() + m11() * dist.y()).
   ```
   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;
   ```
   Returns: vector_2d((m00() * dist.x() + m10() * dist.y()) + m20(), (m01() * dist.x() +
   m11() * dist.y()) + m21()).
   ```
9.7 matrix_2d member operators

\[
\text{matrix}_2d\&\ \text{operator*=}(\text{const} \ \text{matrix}_2d\& \ \text{rhs}) \ \text{noexcept};
\]

1 \hspace{1em} \text{Effects: } \ast\text{this} = \ast\text{this} \ast \text{rhs}
2 \hspace{1em} \text{Returns: } \ast\text{this}

9.8 matrix_2d non-member operators

\[
\text{matrix}_2d \ \text{operator*}(\text{const} \ \text{matrix}_2d\& \ \text{lhs}, \ \text{const} \ \text{matrix}_2d\& \ \text{rhs}) \ \text{noexcept};
\]

1 \hspace{1em} \text{Returns: } \text{matrix}_2d\{
\begin{align*}
\text{lhs.m00()} \ast \text{rhs.m00()} + \text{lhs.m01()} \ast \text{rhs.m10}(), \\
\text{lhs.m00()} \ast \text{rhs.m01()} + \text{lhs.m01()} \ast \text{rhs.m11}(), \\
\text{lhs.m10()} \ast \text{rhs.m00()} + \text{lhs.m11()} \ast \text{rhs.m10}(), \\
\text{lhs.m10()} \ast \text{rhs.m01()} + \text{lhs.m11()} \ast \text{rhs.m11}(), \\
\text{lhs.m20()} \ast \text{rhs.m00()} + \text{lhs.m21()} \ast \text{rhs.m10}() + \text{lhs.m20}(), \\
\text{lhs.m20()} \ast \text{rhs.m01()} + \text{lhs.m21()} \ast \text{rhs.m11}() + \text{lhs.m21}()
\end{align*}
\}
\]

2 \hspace{1em} \text{bool } \text{operator==}(\text{const} \ \text{matrix}_2d\& \ \text{lhs}, \ \text{const} \ \text{matrix}_2d\& \ \text{rhs}) \ \text{noexcept};
3 \hspace{1em} \text{Returns: } \text{lhs.m00()} == \text{rhs.m00}() \&\& \text{lhs.m01()} == \text{rhs.m01}() \&\& \text{lhs.m10()} == \text{rhs.m10}() \&\& \text{lhs.m11()} == \text{rhs.m11}() \&\& \text{lhs.m20()} == \text{rhs.m20}() \&\& \text{lhs.m21()} == \text{rhs.m21}().

3 \hspace{1em} \text{bool } \text{operator!=(const} \ \text{matrix}_2d\& \ \text{lhs, const} \ \text{matrix}_2d\& \ \text{rhs}) \ \text{noexcept};
3 \hspace{1em} \text{Returns: } !(\text{lhs} == \text{rhs}).
10 Paths

Paths define geometric objects which can be stroked (Table 21), filled, masked, and used to define or modify
a Clip Area (Table 20).

Paths are created using a path_factory object.

They provide vector graphics functionality and as such are particularly useful in situations where an applica-
tion is intended to run on a variety of platforms whose output devices (13.12.2) span a large gamut of
sizes, both physical and in terms of pixel dimensions.

10.1 Path geometries

10.1.1 Overview of path geometries

Path geometries are most easily formed using a path_factory object.

They may also be formed by directly creating and manipulating a vector<path_data_item> object.

A path geometry may contain degenerate path segments.

There are special rules concerning the rendering of degenerate path segments. As such they shall be added
to a path geometry when requested and shall not be removed from a path geometry when processing it.

A path object is an immutable resource wrapper containing a path geometry graphics resource (10.4).

10.1.2 Processing path geometries

10.1.2 describes how a properly formed vector<path_data_item> object shall be interpreted by an imple-
mentation in order to convert it into a path geometry graphics resource.

The native_geometry_collection class, described below, is used for expository purposes only. It provides
a hypothetical interface for forming a path geometry graphics resource object. It is not normative.

The native_geometry_collection class has the following state data, the types of which are unspecified:

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Point</td>
<td>The start point for a path segment that is added to the Current Path Geometry.</td>
</tr>
<tr>
<td>Close Point</td>
<td>The start point for the initial path segment in the Current Path Geometry.</td>
</tr>
<tr>
<td>Current Path Geometry</td>
<td>The path geometry to which path segments are added.</td>
</tr>
</tbody>
</table>
| Collection            | The collection of all path geometries added to the
native_geometry_collection object. A new path geometry that is added to the collection is added to the end of the collection. |

// This class is exposition only
class native_geometry_collection {
pUBLIC:
    void current_point(const vector_2d& pt) noexcept;
    void close_point(const vector_2d& pt) noexcept;

§ 10.1.2
void line_to(const vector_2d& pt) noexcept;
void curve_to(const vector_2d& control1, const vector_2d& control2,
const vector_2d& endPt) noexcept;
void close_path() noexcept;
};

void current_point(const vector_2d& pt) noexcept;

Effects: If the last member function called was not current_point or if the Collection contains no
path geometries, a new path geometry is created, added to the Collection, and set as the Current Path
Geometry.

If a new path geometry is created and the Collection contained at least one path geometry prior to
this member function being called, then unless the last member function called was close_path, the
previous Current Path Geometry shall be an open path geometry.

Sets pt as the Current Point.

void close_point(const vector_2d& pt) noexcept;

Requires: There is a Current Path Geometry.

Effects: Sets pt as the Close Point.

void line_to(const vector_2d& pt) noexcept;

Requires: There is a Current Path Geometry.

Effects: Creates a line segment from the Current Point to pt and adds it to the Current Path Geometry.

void curve_to(const vector_2d& cpt1, const vector_2d& cpt2,
const vector_2d& endPt) noexcept;

Requires: There is a Current Path Geometry.

Effects: Creates a cubic Bézier curve from the Current Point to endPt using cpt1 as the first control
point and cpt2 as the second control point and adds it to the Current Path Geometry.

void close_path() noexcept;

Requires: There is a Current Path Geometry.

Effects: Creates a line segment from the Current Point to the Close Point.

The Current Path Geometry becomes a closed path geometry.

Note: A path geometry graphics resource that only supports rendering triangles is possible. The triangles
would be used to form lines and to approximate curves. This description assumes the existence of a path
geometry graphics resource that performs those actions where needed. —end note

The following code shows how to properly process vector<path_data_item> p and store the results into
native_geometry_collection n:

const double pi = 3.1415926535897932384626433832795;
cost double halfpi = 1.57079632679489661923132169163985;
cost double twopi = 6.283185307179586476925286766559;
matrix_2d m;
vector_2d origin;
vector_2d currentPoint;
bool hasCurrentPoint = false;
vector_2d closePoint;
for (const auto& item : p) {
switch(item.type()) {
case path_data_type::move_to:
{
    currentPoint = item.get<path_data_item::move_to>().to();
    auto pt = m.transform_point(currentPoint - origin) + origin;
    n.current_point(pt);
    hasCurrentPoint = true;
    closePoint = pt
    n.close_point(pt);
} break;
case path_data_type::line_to:
{
    currentPoint = item.get<path_data_item::line_to>().to();
    auto pt = m.transform_point(currentPoint - origin) + origin;
    if (hasCurrentPoint) {
        n.line_to(pt);
    }
    else {
        n.current_point(pt);
        hasCurrentPoint = true;
        closePoint = pt;
        n.close_point(pt);
    }
} break;
case path_data_type::curve_to:
{
    auto cd = item.get<path_data_item::curve_to>();
    auto pt1 = m.transform_point(cd.control_point_1() - origin) + origin;
    auto pt2 = m.transform_point(cd.control_point_2() - origin) + origin;
    auto pt3 = m.transform_point(cd.end_point() - origin) + origin;
    if (!hasCurrentPoint) {
        currentPoint = cd.control_point_1();
        n.current_point(pt1);
        hasCurrentPoint = true;
        closePoint = pt1;
        n.close_point(pt1);
    }
    n.curve_to(pt1, pt2, pt3);
    currentPoint = cd.end_point();
} break;
case path_data_type::new_sub_path:
{
    hasCurrentPoint = false;
} break;
case path_data_type::close_path:
{
    if (!hasCurrentPoint) {
        break;
    }
    n.close_path();
    n.current_point(closePoint);
    // Invert can error so use correct overload; here is the throw version.
    auto invM = matrix_2d{m}.invert();
    // Need the untransformed value for currentPoint.
currentPoint = invM.transform_point(closePoint - origin) + origin;
} break;
case path_data_type::rel_move_to:
{
    // If !hasCurrentPoint, error is io2d_error::no_current_point;
    currentPoint = item.get<path_data_item::rel_move_to>().to() + currentPoint;
    auto pt = m.transform_point(currentPoint - origin) + origin;
    n.current_point(pt);
    hasCurrentPoint = true;
    closePoint = pt
    n.close_point(pt);
} break;
case path_data_type::rel_line_to:
{
    // If !hasCurrentPoint, error is io2d_error::no_current_point;
    currentPoint = item.get<path_data_item::rel_line_to>().to() + currentPoint;
    auto pt = m.transform_point(currentPoint - origin) + origin;
    n.line_to(pt);
} break;
case path_data_type::rel_curve_to:
{
    // If !hasCurrentPoint, error is io2d_error::no_current_point;
    auto cd = item.get<path_data_item::rel_curve_to>();
    auto pt1 = m.transform_point(cd.control_point_1() + currentPoint - origin) + origin;
    auto pt2 = m.transform_point(cd.control_point_2() + currentPoint - origin) + origin;
    auto pt3 = m.transform_point(cd.end_point() + currentPoint - origin) + origin;
    n.curve_to(pt1, pt2, pt3);
    currentPoint = cd.end_point() + currentPoint;
} break;
case path_data_type::arc:
{
    auto ad = item.get<path_data_item::arc>();
    auto ctr = ad.center();
    auto rad = ad.radius();
    auto ang1 = ad.angle_1();
    auto ang2 = ad.angle_2();
    while(ang2 < ang1) {
        ang2 += twopi;
    }
    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;
  n.line_to(pt);
} else {
  currentPoint = startPt;
  auto pt = m.transform_point(currentPoint - origin) + origin;
  n.current_point(pt);
  hasCurrentPoint = true;
  closePt = pt;
  n.close_point(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;
  n.curve_to(cpt1, cpt2, cpt3);
  currTheta += theta;
}
} break;
case path_data_type::arc_negative:
{
  auto ad = item.get<path_data_item::arc_negative>();
  auto ctr = ad.center();
  auto rad = ad.radius();
  auto ang1 = ad.angle_1();
  auto ang2 = ad.angle_2();
  while(ang2 > ang1) {
    ang2 -= twopi;
  }
  vector_2d pt0, pt1, pt2, pt3;
  § 10.1.2 53
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.y((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;
    n.line_to(pt);
} else {
    currentPoint = startPt;
    auto pt = m.transform_point(currentPoint - origin) + origin;
    n.current_point(pt);
    hasCurrentPoint = true;
    closePt = pt;
    n.close_point(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;

§ 10.1.2 54
cpt3 = m.transform_point(cpt3 - origin) + origin;
n.curve_to(cpt1, cpt2, cpt3);
currTheta -= theta;
}
} break;
case path_data_type::change_matrix:
{
    m = item.get<path_data_item::change_matrix>().matrix();
} break;
case path_data_type::change_origin:
{
    origin = item.get<path_data_item::change_origin>().origin();
} break;
}

10.2 Enum class path_data_type

10.2.1 path_data_type Summary

The path_data_type enum class specifies the polymorphic type of a path_data object. See Table 3 for the meaning of each path_data_type enumerator.

10.2.2 path_data_type Synopsis

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

10.2.3 path_data_type Enumerators

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>move_to</td>
<td>The object is of type move_to.</td>
</tr>
<tr>
<td>line_to</td>
<td>The object is of type line_to.</td>
</tr>
<tr>
<td>curve_to</td>
<td>The object is of type curve_to.</td>
</tr>
<tr>
<td>new_sub_path</td>
<td>The object is of type new_sub_path.</td>
</tr>
<tr>
<td>close_path</td>
<td>The object is of type close_path.</td>
</tr>
<tr>
<td>rel_move_to</td>
<td>The object is of type rel_move_to.</td>
</tr>
<tr>
<td>rel_line_to</td>
<td>The object is of type rel_line_to.</td>
</tr>
<tr>
<td>rel_curve_to</td>
<td>The object is of type rel_curve_to.</td>
</tr>
</tbody>
</table>
Table 3 — path_data_type enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>arc</td>
<td>The object is of type arc.</td>
</tr>
<tr>
<td>arc_negative</td>
<td>The object is of type arc_negative.</td>
</tr>
<tr>
<td>change_matrix</td>
<td>The object is of type change_matrix.</td>
</tr>
<tr>
<td>change_origin</td>
<td>The object is of type change_origin.</td>
</tr>
</tbody>
</table>

10.3 Class path_data_item

10.3.1 path_data_item synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class path_data_item {

    public:
    class path_data;
    class arc;
    class arc_negative;
    class change_matrix;
    class change_origin;
    class close_path;
    class curve_to;
    class line_to;
    class move_to;
    class new_sub_path;
    class rel_curve_to;
    class rel_line_to;
    class rel_move_to;

    // 10.3.3, construct/copy/move/destroy:
    path_data_item() noexcept;
    path_data_item(const path_data_item& other) noexcept;
    path_data_item& operator=(const path_data_item& other) noexcept;
    path_data_item(path_data_item&& other) noexcept;
    path_data_item& operator=(path_data_item&& other) noexcept;
    path_data_item(const path_data_item::arc& value) noexcept;
    path_data_item(const path_data_item::arc_negative& value) noexcept;
    path_data_item(const path_data_item::change_matrix& value) noexcept;
    path_data_item(const path_data_item::change_origin& value) noexcept;
    path_data_item(const path_data_item::close_path& value) noexcept;
    path_data_item(const path_data_item::curve_to& value) noexcept;
    path_data_item(const path_data_item::line_to& value) noexcept;
    path_data_item(const path_data_item::move_to& value) noexcept;
    path_data_item(const path_data_item::new_sub_path& value) noexcept;
    path_data_item(const path_data_item::rel_curve_to& value) noexcept;
    path_data_item(const path_data_item::rel_line_to& value) noexcept;
    path_data_item(const path_data_item::rel_move_to& value) noexcept;

    // 10.3.4, modifiers:
    void assign(const path_data_item::arc& value) noexcept;
    void assign(const path_data_item::arc_negative& value) noexcept;
    void assign(const path_data_item::change_matrix& value) noexcept;
    void assign(const path_data_item::change_origin& value) noexcept;
    void assign(const path_data_item::close_path& value) noexcept;
    void assign(const path_data_item::curve_to& value) noexcept;

§ 10.3.1 56
void assign(const path_data_item::rel_curve_to& value) noexcept;
void assign(const path_data_item::new_sub_path& value) noexcept;
void assign(const path_data_item::line_to& value) noexcept;
void assign(const path_data_item::move_to& value) noexcept;
void assign(const path_data_item::rel_line_to& value) noexcept;
void assign(const path_data_item::rel_move_to& value) noexcept;

// 10.3.5, observers:
bool has_data() const noexcept;
path_data_type type() const;
path_data_type type(error_code& ec) const noexcept;

template <class T>
T get() const;
template <class T>
T get(error_code& ec) const noexcept;

private:
bool _Has_data;       // exposition only
union {
    struct {
        double centerX;
        double centerY;
        double radius;
        double angle1;
        double angle2;
    } arc;
    struct {
        double m00;
        double m01;
        double m10;
        double m11;
        double m20;
        double m21;
    } matrix;
    struct {
        double cpt1x;
        double cpt1y;
        double cpt2x;
        double cpt2y;
        double eptx;
        double epty;
    } curve;
    struct {
        double x;
        double y;
    } point;
} _Data;            // exposition only

path_data_type _Type;    // exposition only
};
10.3.2 path_data_item Description

The class path_data_item describes an opaque container capable of storing and retrieving an object of a type derived from path_data.

10.3.3 path_data_item constructors and assignment operators

path_data_item() noexcept;

Effects: Constructs an object of type path_data_item.
Postconditions: _Has_data == false.

path_data_item(const path_data_item::arc& value) noexcept;

Effects: Constructs an object of type path_data_item.
Postconditions: _Has_data == true.
_Type == path_data_type::arc.
_Data.arc.centerX == value.center().x().
_Data.arc.centerY == value.center().y().
_Data.arc.radius == value.radius().
_Data.arc.angle1 == value.angle_1().
_Data.arc.angle2 == value.angle_2().

path_data_item(const path_data_item::arc_negative& value) noexcept;

Effects: Constructs an object of type path_data_item.
Postconditions: _Has_data == true.
_Type == path_data_type::arc_negative.
_Data.arc.centerX == value.center().x().
_Data.arc.centerY == value.center().y().
_Data.arc.radius == value.radius().
_Data.arc.angle1 == value.angle_1().
_Data.arc.angle2 == value.angle_2().

path_data_item(const path_data_item::change_matrix& value) noexcept;

Effects: Constructs an object of type path_data_item.
Postconditions: _Has_data == true.
_Type == path_data_type::change_matrix.
_Data.matrix.m00 == value.matrix().m00().
_Data.matrix.m01 == value.matrix().m01().
_Data.matrix.m10 == value.matrix().m10().
_Data.matrix.m11 == value.matrix().m11().
_Data.matrix.m20 == value.matrix().m20().
_Data.matrix.m21 == value.matrix().m21().

path_data_item(const path_data_item::change_origin& value) noexcept;

§ 10.3.3
Effects: Constructs an object of type path_data_item.

Postconditions: _Has_data == true.
_Type == path_data_type::change_origin.
_Data.point.x == value.origin().x().
_Data.point.y == value.origin().y().

path_data_item(const path_data_item::close_path& value) noexcept;

Effects: Constructs an object of type path_data_item.

Postconditions: _Has_data == true.
_Type == path_data_type::close_path.

path_data_item(const path_data_item::curve_to& value) noexcept;

Effects: Constructs an object of type path_data_item.

Postconditions: _Has_data == true.
_Type == path_data_type::curve_to.
_Data.curve.cpt1x == value.control_point_1().x().
_Data.curve.cpt1y == value.control_point_1().y().
_Data.curve.cpt2x == value.control_point_2().x().
_Data.curve.cpt2y == value.control_point_2().y().
_Data.curve.eptx == value.end_point().x().
_Data.curve.epty == value.end_point().y().

path_data_item(const path_data_item::rel_curve_to& value) noexcept;

Effects: Constructs an object of type path_data_item.

Postconditions: _Has_data == true.
_Type == path_data_type::rel_curve_to.
_Data.curve.cpt1x == value.control_point_1().x().
_Data.curve.cpt1y == value.control_point_1().y().
_Data.curve.cpt2x == value.control_point_2().x().
_Data.curve.cpt2y == value.control_point_2().y().
_Data.curve.eptx == value.end_point().x().
_Data.curve.epty == value.end_point().y().

path_data_item(const path_data_item::new_sub_path& value) noexcept;

Effects: Constructs an object of type path_data_item.

Postconditions: _Has_data == true.
_Type == path_data_type::new_sub_path.

path_data_item(const path_data_item::line_to& value) noexcept;
Effects: Constructs an object of type `path_data_item`.

Postconditions: _Has_data == true.
_Type == path_data_type::line_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

path_data_item(const path_data_item::move_to& value) noexcept;

Effects: Constructs an object of type `path_data_item`.

Postconditions: _Has_data == true.
_Type == path_data_type::move_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

path_data_item(const path_data_item::rel_line_to& value) noexcept;

Effects: Constructs an object of type `path_data_item`.

Postconditions: _Has_data == true.
_Type == path_data_type::rel_line_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

path_data_item(const path_data_item::rel_move_to& value) noexcept;

Effects: Constructs an object of type `path_data_item`.

Postconditions: _Has_data == true.
_Type == path_data_type::rel_move_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

10.3.4 path_data_item modifiers

void assign(const path_data_item::arc& value) noexcept;

Postconditions: _Has_data == true.
_Type == path_data_type::arc.
_Data.arc.centerX == value.center().x().
_Data.arc.centerY == value.center().y().
_Data.arc.radius == value.radius().
_Data.arc.angle1 == value.angle_1().
_Data.arc.angle2 == value.angle_2().

void assign(const path_data_item::arc_negative& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::arc_negative.
_Data.arc.centerX == value.center().x().
_Data.arc.centerY == value.center().y().
_Data.arc.radius == value.radius().
_Data.arc.angle1 == value.angle_1().
_Data.arc.angle2 == value.angle_2().

void assign(const path_data_item::change_matrix& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::change_matrix.
_Data.matrix.m00 == value.matrix().m00().
_Data.matrix.m01 == value.matrix().m01().
_Data.matrix.m10 == value.matrix().m10().
_Data.matrix.m11 == value.matrix().m11().
_Data.matrix.m20 == value.matrix().m20().
_Data.matrix.m21 == value.matrix().m21().

void assign(const path_data_item::change_origin& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::change_origin.
_Data.point.x == value.origin().x().
_Data.point.y == value.origin().y().

void assign(const path_data_item::close_path& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::close_path.

void assign(const path_data_item::curve_to& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::curve_to.
_Data.curvecpt1x == value.control_point_1().x().
_Data.curvecpt1y == value.control_point_1().y().
_Data.curvecpt2x == value.control_point_2().x().
_Data.curvecpt2y == value.control_point_2().y().
_Data.curveeptx == value.end_point().x().
_Data.curveepty == value.end_point().y().

void assign(const path_data_item::rel_curve_to& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::rel_curve_to.
_Data.curve.cpt1x == value.control_point_1().x().
_Data.curve.cpt1y == value.control_point_1().y().
_Data.curve.cpt2x == value.control_point_2().x().
_Data.curve.cpt2y == value.control_point_2().y().
_Data.curve.eptx == value.end_point().x().
_Data.curve.epty == value.end_point().y().

void assign(const path_data_item::new_sub_path& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::new_sub_path.

void assign(const path_data_item::line_to& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::line_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

void assign(const path_data_item::move_to& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::move_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

void assign(const path_data_item::rel_line_to& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::rel_line_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().

void assign(const path_data_item::rel_move_to& value) noexcept;
Postconditions: _Has_data == true.
_Type == path_data_type::rel_move_to.
_Data.point.x == value.to().x().
_Data.point.y == value.to().y().
10.3.5  path_data_item observers

bool has_data() const noexcept;

1  Returns: _Has_data.

path_data_type type() const;
path_data_type type(error_code& ec) const noexcept;

2  Returns: _Type.
3  Throws: As specified in Error reporting (3).
4  Error conditions: errc::operation_not_permitted if !this->has_data().

10.3.6  path_data_item member function templates

The specializations of the path_data_item::get member function template are specified at 10.3.20.

10.3.7  Class path_data_item::path_data

The class path_data_item::path_data serves as an abstract base class for classes that describe operations performed on path geometries.

10.3.7.1  path_data_item::path_data synopsis

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

class path_data_item::path_data {
    public:
        // 10.3.7.2, construct/copy/move/destroy:
        path_data() noexcept;  
        path_data(const path_data& other) noexcept;
        path_data& operator=(const path_data& other) noexcept;
        path_data(path_data&& other) noexcept;
        path_data& operator=(path_data&& other) noexcept;
        virtual ~path_data() noexcept;

        // 10.3.7.3, observers:
        virtual path_data_type type() const noexcept = 0;
    }
};
}
}}

10.3.7.2  path_data_item::path_data constructors and assignment operators

virtual ~path_data() noexcept;

1  Effects: Destroys an object of class path_data_item::path_data.

10.3.7.3  path_data_item::path_data observers

virtual path_data_type type() const noexcept = 0;

1  Returns: The path_data_type of the path_data-derived object.
2  Note: This is used for casting to the correct type when iterating through a collection of path_data objects.
10.3.8 Class path_data_item::arc

The class `path_data_item::arc` describes an operation on a path geometry collection.

This operation creates a circular arc with clockwise rotation.

The unit for the values passed to and returned by `path_data_item::arc::angle_1` and `path_data_item::arc::angle_2` is the radian.

The arc's start point is `vector_2d{ *this.radius() * cos(*this.angle_1(), -(*this.radius() * -sin(*this.angle_1()))) } + *this.center()`.

Its end point is `vector_2d{ *this.radius() * cos(*this.angle_2(), -(*this.radius() * -sin(*this.angle_2()))) } + *this.center()`.

If the current path geometry has a current point, a line is created from the current point to the start point before this arc operation is processed. Otherwise the start point is set as the current point and last-move-to point of the current path geometry.

The arc rotates around the point returned by `*this.center()`.

The arc begins at its start point and proceeds clockwise until it reaches its end point.

The current point is set be to the arc’s end point at the end of this operation.

For purposes of determining whether a point is on the arc, if the value returned by `*this.angle_2()` is less than the value returned by `*this.angle_1()` then the value returned by `*this.angle_2()` shall be continuously incremented by $2\pi$ until it is greater than the value returned by `*this.angle_1()`.

10.3.8.1 path_data_item::arc synopsis

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

    class path_data_item::arc : public path_data_item::path_data {
    public:
        // 10.3.8.2, construct/copy/move/destroy:
        arc() noexcept;
        arc(const arc&) noexcept;
        path_data_item::arc& operator=(const arc&) noexcept;
        arc(arc&&) noexcept;
        path_data_item::arc& operator=(arc&&) noexcept;
        arc(const vector_2d& ctr, double rad, double angle1, double angle2) noexcept;

        // 10.3.8.3, modifiers:
        void center(const vector_2d& value) noexcept;
        void radius(double value) noexcept;
        void angle_1(double radians) noexcept;
        void angle_2(double radians) noexcept;

        // 10.3.8.4, observers:
        vector_2d center() const noexcept;
        double radius() const noexcept;
        double angle_1() const noexcept;
        double angle_2() const noexcept;
        virtual path_data_type type() const noexcept override;

    private:
        vector_2d _Center; // exposition only
        double _Radius; // exposition only
        double _Angle_1; // exposition only
        double _Angle_2; // exposition only
    };

§ 10.3.8.1
10.3.8.2 path_data_item::arc constructors and assignment operators

arc() noexcept;

Effects: Constructs an object of type path_data_item::arc.

Postconditions: _Center == vector_2d(0.0, 0.0).
_Radius == 0.0.
_Angle_1 == 0.0.
_Angle_2 == 0.0.

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

Effects: Constructs an object of type path_data_item::arc.

Postconditions: _Center == ctr.
_Radius == rad.
_Angle_1 == angle1.
_Angle_2 == angle2.

10.3.8.3 path_data_item::arc modifiers

void center(const vector_2d& value) noexcept;

Postconditions: _Center == value.

void radius(double value) noexcept;

Postconditions: _Radius == value.

void angle_1(double value) noexcept;

Postconditions: _Angle_1 == value.

void angle_2(double value) noexcept;

Postconditions: _Angle_2 == value.

10.3.8.4 path_data_item::arc observers

vector_2d center() const noexcept;

Returns: _Center.

double radius() const noexcept;

Returns: _Radius.

double angle_1() const noexcept;

Returns: _Angle_1.

double angle_2() const noexcept;

Returns: _Angle_2.

virtual path_data_type type() const noexcept override;

Returns: path_data_type::arc.
10.3.9 Class path_data_item::arc_negative

The class path_data_item::arc_negative describes an operation on a path geometry collection.

This operation creates a circular arc with counterclockwise rotation.

The unit for the values passed to and returned by path_data_item::arc_negative::angle_1 and path_data_item::arc_negative::angle_2 is the radian.

The arc’s start point is vector_2d{ *this.radius() * cos(*this.angle_1()), *this.radius() * -sin(*this.angle_1()) } + *this.center().

Its end point is vector_2d{ *this.radius() * cos(*this.angle_2()), *this.radius() * -sin(*this.angle_2()) } + *this.center().

If the current path geometry has a current point, a line is created from the current point to the start point before this arc operation is processed. Otherwise the start point is set as the current point and last-move-to point of the current path geometry.

The arc rotates around the point returned by *this.center().

The arc begins at its start point and proceeds counterclockwise until it reaches its end point.

The current point is set be to the arc’s end point at the end of this operation.

For purposes of determining whether a point is on the arc, if the value returned by *this.angle_2() is greater than the value returned by *this.angle_1() then the value returned by *this.angle_2() shall be continuously decremented by 2π until it is less than the value returned by *this.angle_1().

10.3.9.1 path_data_item::arc_negative synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 { class path_data_item::arc_negative : public path_data_item::path_data {
    public:
        // 10.3.9.2, construct/copy/move/destroy:
        arc_negative() noexcept;
        arc_negative(const arc_negative&) noexcept;
        path_data_item::arc_negative& operator=(const arc_negative&) noexcept;
        arc_negative(arc_negative&&) noexcept;
        path_data_item::arc_negative& operator=(arc_negative&&) noexcept;
        arc_negative(const vector_2d& ctr, double rad, double angle1, double angle2) noexcept;

        // 10.3.9.3, modifiers:
        void center(const vector_2d& value) noexcept;
        void radius(double value) noexcept;
        void angle_1(double radians) noexcept;
        void angle_2(double radians) noexcept;

        // 10.3.9.4, observers:
        vector_2d center() const noexcept;
        double radius() const noexcept;
        double angle_1() const noexcept;
        double angle_2() const noexcept;
        virtual path_data_type type() const noexcept override;

    private:
        vector_2d _Center; // exposition only
        double _Radius; // exposition only
        double _Angle_1; // exposition only
        double _Angle_2; // exposition only

§ 10.3.9.1
10.3.9.2 path_data_item::arc_negative constructors and assignment operators

arc_negative() noexcept;
Effects: Constructs an object of type path_data_item::arc_negative.
Postconditions: _Center == vector_2d(0.0, 0.0).
_Radius == 0.0.
_Angle_1 == 0.0.
_Angle_2 == 0.0.

arc_negative(const vector_2d& ctr, double rad, double angle1,
  double angle2) noexcept;
Effects: Constructs an object of type path_data_item::arc_negative.
Postconditions: _Center == ctr.
_Radius == rad.
_Angle_1 == angle1.
_Angle_2 == angle2.

10.3.9.3 path_data_item::arc_negative modifiers

void center(const vector_2d& value) noexcept;
Postconditions: _Center == value.

void radius(double value) noexcept;
Postconditions: _Radius == value.

void angle_1(double value) noexcept;
Postconditions: _Angle_1 == value.

void angle_2(double value) noexcept;
Postconditions: _Angle_2 == value.

10.3.9.4 path_data_item::arc_negative observers

vector_2d center() const noexcept;
Returns: _Center.

double radius() const noexcept;
Returns: _Radius.

double angle_1() const noexcept;
Returns: _Angle_1.

double angle_2() const noexcept;
Returns: _Angle_2.

virtual path_data_type type() const noexcept override;
Returns: path_data_type::arc_negative.
10.3.10 Class path_data_item::close_path

The class path_data_item::close_path describes an operation on a path geometry collection.

1 If the current path geometry has a current point, this operation creates a line from the current point to the last-move-to point. It then starts a new path geometry and sets its current point and last-move-to point to the value of the previous path geometry’s last-move-to point.

2 If there is no current point, then this operation does nothing. [Note: Because this operation does nothing if there is no current point, there is no need to track whether or not a path geometry has a valid last-move-to point. This operation is the only operation that uses the last-move-to point and all operations that establish a current point for a path geometry also establish a valid last-move-to point for that path geometry. —end note]

10.3.10.1 path_data_item::close_path synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class path_data_item::close_path : public path_data_item::path_data {
    public:
        // construct/copy/move/destroy:
        close_path() noexcept;
        close_path(const close_path&) noexcept;
        path_data_item::close_path& operator=(const close_path&) noexcept;
        close_path(close_path&&) noexcept;
        path_data_item::close_path& operator=(close_path&&) noexcept;

        // 10.3.10.2, observers:
        virtual path_data_type type() const noexcept override;
    } } } }

10.3.10.2 path_data_item::close_path observers

virtual path_data_type type() const noexcept override;

Returns: path_data_type::close_path.

10.3.11 Class path_data_item::change_matrix

The class path_data_item::change_matrix describes an operation on a path geometry collection.

1 This operation changes the transformation matrix for a path geometry collection to be the value returned by *this.matrix(). As shown in 10.1.2, the new transformation matrix does not affect any operations that came before this operation. It is only used in processing operations that come after it. It continues to be used until another path_data_item::change_matrix object is encountered or the end of the path geometry collection is reached.

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class path_data_item::change_matrix : public path_data_item::path_data {
    public:
        // 10.3.11.2, construct/copy/move/destroy:
        change_matrix() noexcept;
        change_matrix(const change_matrix&) noexcept;
        path_data_item::change_matrix& operator=(const change_matrix&) noexcept;
        change_matrix(change_matrix&&) noexcept;
        path_data_item::change_matrix& operator=(change_matrix&&) noexcept;
        explicit change_matrix(const matrix_2d& m) noexcept;

§ 10.3.11.1 68
10.3.11.3, modifiers:
void matrix(const matrix_2d& value) noexcept;

10.3.11.4, observers:
matrix_2d matrix() const noexcept;
virtual path_data_type type() const noexcept override;

private:
  matrix_2d _Matrix; // exposition only
};
} } } }

10.3.11.2 path_data_item::change_matrix constructors and assignment operators
[ pathdataitem.changematrix.cons ]

change_matrix() noexcept;
1 Effects: Constructs an object of type path_data_item::change_matrix.
2 Postconditions: _Matrix == matrix_2d{}.

explicit change_matrix(const matrix_2d& m) noexcept;
3 Effects: Constructs an object of type path_data_item::change_matrix.
4 Postconditions: _Matrix == m.

10.3.11.3 path_data_item::change_matrix modifiers [ pathdataitem.changematrix.modifiers ]

void matrix(const matrix_2d& value) noexcept;
1 Postconditions: _Matrix == value.

10.3.11.4 path_data_item::change_matrix observers [ pathdataitem.changematrix.observers ]

matrix_2d matrix() const noexcept;
1 Returns: _Matrix.

virtual path_data_type type() const noexcept override;
2 Returns: path_data_type::change_matrix.

10.3.12 Class path_data_item::change_origin [ pathdataitem.changeorigin ]
The class path_data_item::change_origin describes an operation on a path geometry collection.
2 This operation changes the origin point for a path geometry collection to be the value returned by *this.origin().
As shown in 10.1.2, the new origin point does not affect any operations that came before this operation. It is only used in processing operations that come after it. It continues to be used until another path_data_item::change_origin object is encountered or the end of the path geometry collection is reached.

10.3.12.1 path_data_item::change_origin synopsis [ pathdataitem.changeorigin.synopsis ]

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
class path_data_item::change_origin : public path_data_item::path_data {
  public:
    // 10.3.12.2, construct/copy/move/destroy:
    change_origin() noexcept;
    change_origin(const change_origin&) noexcept;
    path_data_item::change_origin& operator=(const change_origin&) noexcept;

§ 10.3.12.1
change_origin(change_origin&&) noexcept;
path_data_item::change_origin& operator=(change_origin&&) noexcept;
explicit change_origin(const vector_2d& pt) noexcept;

// 10.3.12.3, modifiers:
void origin(const vector_2d& value) noexcept;

// 10.3.12.4, observers:
vector_2d origin() const noexcept;
virtual path_data_type type() const noexcept override;

private:
    vector_2d _Data;  // exposition only
};

10.3.12.2 path_data_item::change_origin constructors and assignment operators
[pathdataitem.changeorigin.cons]

change_origin() noexcept;
1 Effects: Constructs an object of type path_data_item::change_origin.
2 Postconditions: _Data == vector_2d(0.0, 0.0).
explicit change_origin(const vector_2d& pt) noexcept;
3 Effects: Constructs an object of type path_data_item::change_origin.
4 Postconditions: _Data == pt.

10.3.12.3 path_data_item::change_origin modifiers [pathdataitem.changeorigin.modifiers]

void origin(const vector_2d& value) noexcept;
1 Postconditions: _Data == value.

10.3.12.4 change_origin observers [pathdataitem.changeorigin.observers]

vector_2d origin() const noexcept;
1 Returns: _Data.
virtual path_data_type type() const noexcept override;
2 Returns: path_data_type::change_origin.

10.3.13 Class path_data_item::curve_to [pathdataitem.curveto]

1 The class path_data_item::curve_to describes an operation on a path geometry collection.
2 If the current path geometry has no current point, then this operation behaves exactly as if this object was preceded by a path_data_item::move_to object constructed with the value returned by *this.control_point_1() as its argument.
3 This operation creates a cubic Bézier curve from the current point to the point returned by *this.end_point(), with the first control point being the point returned by *this.control_point_1() and the second control point being the point returned by *this.control_point_2(). It then sets the current point to be the point returned by *this.end_point().

§ 10.3.13
# 10.3.13.1 path_data_item::curve_to synopsis

```cpp
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class path_data_item::curve_to : public path_data_item::path_data {
        public:
            // 10.3.13.2, construct/copy/move/destroy:
            curve_to() noexcept;
            curve_to(const curve_to&) noexcept;
            path_data_item::curve_to& operator=(const curve_to&) noexcept;
            curve_to(curve_to&&) noexcept;
            path_data_item::curve_to& operator=(curve_to&&) noexcept;
            curve_to(const vector_2d& controlPoint1, const vector_2d& controlPoint2,
                const vector_2d& endPoint) noexcept;

            // 10.3.13.3, modifiers:
            void control_point_1(const vector_2d& value) noexcept;
            void control_point_2(const vector_2d& value) noexcept;
            void end_point(const vector_2d& value) noexcept;

            // 10.3.13.4, observers:
            vector_2d control_point_1() const noexcept;
            vector_2d control_point_2() const noexcept;
            vector_2d end_point() const noexcept;
            virtual path_data_type type() const noexcept override;

        private:
            vector_2d _Control_pt1; // exposition only
            vector_2d _Control_pt2; // exposition only
            vector_2d _End_pt; // exposition only
    }}}
```
void control_point_2(const vector_2d& value) noexcept;

Postconditions: _Control_pt_2 == value.

void end_point(const vector_2d& value) noexcept;

Postconditions: _End_pt == value.

10.3.13.4 path_data_item::curve_to observers

vector_2d control_point_1() const noexcept;

Returns: _Control_pt_1.

vector_2d control_point_2() const noexcept;

Returns: _Control_pt_2.

vector_2d end_point() const noexcept;

Returns: _End_pt.

virtual path_data_type type() const noexcept override;

Returns: path_data_type::curve_to.

10.3.14 Class path_data_item::line_to

The class path_data_item::line_to describes an operation on a path geometry collection.

If the current path geometry has a current point then this operation creates a line from the current point to the point returned by *this.to() and then sets current point to be the point returned by *this.to(). Otherwise, this operation behaves exactly as if this object was a path_data_item::move_to object constructed which was constructed with the value returned by *this.to() as its argument.

10.3.14.1 path_data_item::line_to synopsis

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

class path_data_item::line_to : public path_data_item::path_data {

public:

// 10.3.14.2, construct/copy/move/destroy:
line_to() noexcept;
line_to(const line_to&) noexcept;
path_data_item::line_to& operator=(const line_to&) noexcept;
line_to(line_to&&) noexcept;
path_data_item::line_to& operator=(line_to&&) noexcept;
explicit line_to(const vector_2d& pt) noexcept;

// 10.3.14.3, modifiers:
void to(const vector_2d& pt) noexcept;

// 10.3.14.4, observers:
vector_2d to() const noexcept;
virtual path_data_type type() const noexcept override;

private:

vector_2d _Data; // exposition only
}
};
}}}

§ 10.3.14.1
10.3.14.2  path_data_item::line_to constructors and assignment operators

[pathdataitem.lineto.cons]

```cpp
line_to() noexcept;
```

1

Effects: Constructs an object of type path_data_item::line_to.

2

Postconditions: _Data == vector_2d(0.0, 0.0).

```cpp
explicit line_to(const vector_2d& pt) noexcept;
```

3

Effects: Constructs an object of type path_data_item::line_to.

4

Postconditions: _Data == pt.

10.3.14.3  path_data_item::line_to modifiers

[pathdataitem.lineto.modifiers]

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

1

Postconditions: _Data == pt.

10.3.14.4  path_data_item::line_to observers

[pathdataitem.lineto.observers]

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

1

Returns: _Data.

2

Returns: path_data_type::line_to.

10.3.15  Class path_data_item::move_to

[pathdataitem.moveto]

The class path_data_item::move_to describes an operation on a path geometry collection.

This operation starts a new path geometry and sets its current point and last-move-to point to the value of *this.to().

10.3.15.1  path_data_item::move_to synopsis

[pathdataitem.moveto.synopsis]

```cpp
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class path_data_item::move_to : public path_data_item::path_data {
        public:
            // 10.3.15.2, construct/copy/move/destroy:
            move_to() noexcept;
            move_to(const move_to&) noexcept;
            path_data_item::move_to& operator=(const move_to&) noexcept;
            move_to(move_to&&) noexcept;
            path_data_item::move_to& operator=(move_to&&) noexcept;
            explicit move_to(const vector_2d& pt) noexcept;

            // 10.3.15.3, modifiers:
            void to(const vector_2d& pt) noexcept;

            // 10.3.15.4, observers:
            vector_2d to() const noexcept;
            virtual path_data_type type() const noexcept override;

        private:
            vector_2d _Data; // exposition only
        } } } }
```

§ 10.3.15.1
10.3.15.2 path_data_item::move_to constructors and assignment operators
[pathdataitem.moveto.cons]

move_to() noexcept;

Effects: Constructs an object of type path_data_item::move_to.

Postconditions: _Data == vector_2d(0.0, 0.0).

explicit move_to(const vector_2d& pt) noexcept;

Effects: Constructs an object of type path_data_item::move_to.

Postconditions: _Data == pt.

10.3.15.3 path_data_item::move_to modifiers [pathdataitem.moveto.modifiers]

void to(const vector_2d& pt) noexcept;

Postconditions: _Data == pt.

10.3.15.4 path_data_item::move_to observers [pathdataitem.moveto.observers]

vector_2d to() const noexcept;

Returns: _Data.

virtual path_data_type type() const noexcept override;

Returns: path_data_type::move_to.

10.3.16 Class path_data_item::new_sub_path [pathdataitem.newsubpath]

The class path_data_item::new_sub_path describes an operation on a path geometry collection.

This operation starts a new path geometry. The new path geometry has no current point.

10.3.16.1 path_data_item::new_sub_path synopsis [pathdataitem.newsubpath.synopsis]

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

class path_data_item::new_sub_path : public path_data_item::path_data {

public:

// construct/copy/move/destroy:
new_sub_path() noexcept;
new_sub_path(const new_sub_path&) noexcept;
path_data_item::new_sub_path& operator=(const new_sub_path&) noexcept;
new_sub_path(new_sub_path&&) noexcept;
path_data_item::new_sub_path& operator=(new_sub_path&&) noexcept;

// 10.3.16.2, observers:
virtual path_data_type type() const noexcept override;
}
} } } }

10.3.16.2 path_data_item::new_sub_path observers [pathdataitem.newsubpath.observers]

virtual path_data_type type() const noexcept override;

Returns: path_data_type::new_sub_path.
10.3.17 Class path_data_item::rel_curve_to

The class path_data_item::rel_curve_to describes an operation on a path geometry collection.

This operation creates a cubic Bézier curve from the current point to the point that is the sum of the current point and the point returned by *this.end_point(), with the first control point being the point that is the sum of the current point and the point returned by *this.control_point_1() and the second control point being the point that is the sum of the current point and the point returned by *this.control_point_2(). It then sets the current point to be the point that is the sum of the current point and the point returned by *this.end_point().

If the current path geometry does not have a current point when this operation is requested the path geometry collection is malformed.

10.3.17.1 path_data_item::rel_curve_to synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class path_data_item::rel_curve_to : public path_data_item::path_data {
    public:
        // 10.3.17.2, construct/copy/move/destroy:
        rel_curve_to() noexcept;
        rel_curve_to(const rel_curve_to&) noexcept;
        path_data_item::rel_curve_to& operator=(const rel_curve_to&) noexcept;
        rel_curve_to(rel_curve_to&&) noexcept;
        path_data_item::rel_curve_to& operator=(rel_curve_to&&) noexcept;
        rel_curve_to(const vector_2d& controlPoint1, const vector_2d& controlPoint2,
                     const vector_2d& endPoint) noexcept;

        // 10.3.17.3, modifiers:
        void control_point_1(const vector_2d& value) noexcept;
        void control_point_2(const vector_2d& value) noexcept;
        void end_point(const vector_2d& value) noexcept;

        // 10.3.17.4, observers:
        vector_2d control_point_1() const noexcept;
        vector_2d control_point_2() const noexcept;
        vector_2d end_point() const noexcept;
        virtual path_data_type type() const noexcept override;

    private:
        vector_2d _Control_pt1;   // exposition only
        vector_2d _Control_pt2;   // exposition only
        vector_2d _End_pt;        // exposition only
    }
} } } }

10.3.17.2 path_data_item::rel_curve_to constructors and assignment operators

rel_curve_to() noexcept;

Effects: Constructs an object of type path_data_item::rel_curve_to.

Postconditions: _Control_pt1 == vector_2d(0.0, 0.0).
               _Control_pt2 == vector_2d(0.0, 0.0).
               _End_pt == vector_2d(0.0, 0.0).
rel_curve_to(const vector_2d& controlPoint1, const vector_2d& controlPoint2, const vector_2d& endPoint) noexcept;

Effects: Constructs an object of type path_data_item::rel_curve_to.

Postconditions: _Control_pt1 == controlPoint1.
_Control_pt2 == controlPoint2.
_End_pt == endPoint.

10.3.17.3 path_data_item::rel_curve_to modifiers

void control_point_1(const vector_2d& value) noexcept;

Postconditions: _Control_pt_1 == value.

void control_point_2(const vector_2d& value) noexcept;

Postconditions: _Control_pt_2 == value.

void end_point(const vector_2d& value) noexcept;

Postconditions: _End_pt == value.

10.3.17.4 path_data_item::rel_curve_to observers

vector_2d control_point_1() const noexcept;

Returns: _Control_pt_1.

vector_2d control_point_2() const noexcept;

Returns: _Control_pt_2.

vector_2d end_point() const noexcept;

Returns: _End_pt.

virtual path_data_type type() const noexcept override;

Returns: path_data_type::rel_curve_to.

10.3.18 Class path_data_item::rel_line_to

The class path_data_item::rel_line_to describes an operation on a path geometry collection.

This operation creates a line from the current point to the point that is the sum of the current point and the point returned by *this.to(). It then sets current point to be the sum of the current point and the point returned by *this.to().

If the current path geometry does not have a current point when this operation is requested the path geometry collection is malformed.

10.3.18.1 path_data_item::rel_line_to synopsis

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

class path_data_item::rel_line_to : public path_data_item::path_data {
public:

// 10.3.18.2, construct/copy/move/destroy:
rel_line_to() noexcept;
rel_line_to(const line_to&) noexcept;
path_data_item::rel_line_to& operator=(const line_to&) noexcept;
rel_line_to(line_to&&) noexcept;

§ 10.3.18.1
path_data_item::rel_line_to& operator=(line_to&&) noexcept;
explicit rel_line_to(const vector_2d& pt) noexcept;

// 10.3.18.3, modifiers:
void to(const vector_2d& pt) noexcept;

// 10.3.18.4, observers:
vector_2d to() const noexcept;
virtual path_data_type type() const noexcept override;

private:
vector_2d _Data; // exposition only
};

10.3.18.2 path_data_item::rel_line_to constructors and assignment operators
[pathdataitem.rellineto.cons]

rel_line_to() noexcept;

1 Effects: Constructs an object of type path_data_item::rel_line_to.
2 Postconditions: _Data == vector_2d(0.0, 0.0).

explicit rel_line_to(const vector_2d& pt) noexcept;
3 Effects: Constructs an object of type path_data_item::rel_line_to.
4 Postconditions: _Data == pt.

10.3.18.3 path_data_item::rel_line_to modifiers [pathdataitem.rellineto.modifiers]

void to(const vector_2d& pt) noexcept;
1 Postconditions: _Data == pt.

10.3.18.4 path_data_item::rel_line_to observers [pathdataitem.rellineto.observers]

vector_2d to() const noexcept;
1 Returns: _Data.
2 Returns: path_data_type::rel_line_to.

virtual path_data_type type() const noexcept override;
1 Returns: path_data_type::rel_line_to.
2

10.3.19 Class path_data_item::rel_move_to [pathdataitem.relmoveto]

The class path_data_item::rel_move_to describes an operation on a path geometry collection.

This operation starts a new path geometry and sets its current point and last-move-to point to the point that is the sum of the previous path geometry's current point and the point returned by *this.to().

If the existing path geometry does not have a current point when this operation is requested the path geometry collection is malformed.

10.3.19.1 path_data_item::rel_move_to synopsis [pathdataitem.relmoveto.synopsis]

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

class path_data_item::rel_move_to : public path_data_item::path_data {

public:
1

// 10.3.19.2, construct/copy/move/destroy:

§ 10.3.19.1
10.3.19.2 path_data_item::rel_move_to constructors and assignment operators

rel_move_to() noexcept;
Effects: Constructs an object of type path_data_item::rel_move_to.
Postconditions: _Data == vector_2d(0.0, 0.0).

explicit rel_move_to(const vector_2d& pt) noexcept;
Effects: Constructs an object of type path_data_item::rel_move_to.
Postconditions: _Data == pt.

10.3.19.3 path_data_item::rel_move_to modifiers

void to(const vector_2d& pt) noexcept;
Postconditions: _Data == pt.

10.3.19.4 path_data_item::rel_move_to observers

vector_2d to() const noexcept;
Returns: _Data.

virtual path_data_type type() const noexcept override;
Returns: path_data_type::rel_move_to.

10.3.20 Class path_data_item::get member function template specializations

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
template <>
path_data_item::arc path_data_item::get() const;
template <>
path_data_item::arc path_data_item::get(error_code& ec) const noexcept;

§ 10.3.20.1
template <>
path_data_item::arc_negative path_data_item::get() const;
template <>
path_data_item::arc_negative path_data_item::get(error_code& ec) const
  noexcept;

template <>
inline path_data_item::change_matrix path_data_item::get() const;
template <>
path_data_item::change_matrix path_data_item::get(error_code& ec) const
  noexcept;

template <>
path_data_item::change_origin path_data_item::get() const;
template <>
path_data_item::change_origin path_data_item::get(error_code& ec) const
  noexcept;

template <>
path_data_item::close_path path_data_item::get() const;
template <>
path_data_item::close_path path_data_item::get(error_code& ec) const noexcept;

template <>
path_data_item::curve_to path_data_item::get() const;
template <>
path_data_item::curve_to path_data_item::get(error_code& ec) const noexcept;

template <>
path_data_item::rel_curve_to path_data_item::get() const;
template <>
path_data_item::rel_curve_to path_data_item::get(error_code& ec) const
  noexcept;

template <>
path_data_item::new_sub_path path_data_item::get() const;
template <>
path_data_item::new_sub_path path_data_item::get(error_code& ec) const
  noexcept;

template <>
path_data_item::line_to path_data_item::get() const;
template <>
path_data_item::line_to path_data_item::get(error_code& ec) const noexcept;

template <>
path_data_item::move_to path_data_item::get() const;
template <>
path_data_item::move_to path_data_item::get(error_code& ec) const noexcept;

template <>
path_data_item::rel_line_to path_data_item::get() const;
template <>
path_data_item::rel_line_to path_data_item::get(error_code& ec) const
  noexcept;
\begin{verbatim}
template <>
path_data_item::rel_move_to path_data_item::get() const;
template <>
path_data_item::rel_move_to path_data_item::get(error_code& ec) const
   noexcept;
\}
\}

10.3.20.2 \hspace{1em} \textbf{path_data_item::get specializations} \\
[\texttt{pathdataitem.get.specializations}]

\begin{verbatim}
template <>
path_data_item::arc path_data_item::get() const;
template <>
path_data_item::arc path_data_item::get(error_code& ec) const noexcept;

\hspace{1em} Returns: A copy of the stored \texttt{path_data_item::arc} object.
\hspace{1em} If an error occurs and the function was called with an \texttt{error_code\&} argument, returns \texttt{path_data_item::arc{ }}.
\hspace{1em} \textit{Throws:} As specified in Error reporting (3).
\hspace{1em} \textit{Error conditions:} \texttt{errc::operation_not_permitted} if \texttt{!this->has_data()}. \texttt{errc::invalid_argument} if \texttt{this->type() != path_data_type::arc}.

\begin{verbatim}
template <>
path_data_item::arc_negative path_data_item::get() const;
template <>
path_data_item::arc_negative path_data_item::get(error_code& ec) const noexcept;

\hspace{1em} Returns: A copy of the stored \texttt{path_data_item::arc_negative} object.
\hspace{1em} If an error occurs and the function was called with an \texttt{error_code\&} argument, returns \texttt{path_data_item::arc_negative{ }}.
\hspace{1em} \textit{Throws:} As specified in Error reporting (3).
\hspace{1em} \textit{Error conditions:} \texttt{errc::operation_not_permitted} if \texttt{!this->has_data()}. \texttt{errc::invalid_argument} if \texttt{this->type() != path_data_type::arc_negative}.

\begin{verbatim}
template <>
inline path_data_item::change_matrix path_data_item::get() const;
template <>
path_data_item::change_matrix path_data_item::get(error_code& ec) const
   noexcept;

\hspace{1em} Returns: A copy of the stored \texttt{path_data_item::change_matrix} object.
\hspace{1em} If an error occurs and the function was called with an \texttt{error_code\&} argument, returns \texttt{path_data_item::change_matrix{ }}.
\hspace{1em} \textit{Throws:} As specified in Error reporting (3).
\hspace{1em} \textit{Error conditions:} \texttt{errc::operation_not_permitted} if \texttt{!this->has_data()}. \texttt{errc::invalid_argument} if \texttt{this->type() != path_data_type::change_matrix}.

\end{verbatim}
\end{verbatim}
\end{verbatim}
Returns: A copy of the stored path_data_item::change_origin object.

If an error occurs and the function was called with an error_code& argument, returns path_data_item::change_origin{ }.

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::change_origin.

template <>
path_data_item::close_path path_data_item::get() const;

template <>
path_data_item::close_path path_data_item::get(error_code& ec) const noexcept;

Returns: A copy of the stored path_data_item::close_path object.

If an error occurs and the function was called with an error_code& argument, returns path_data_item::close_path{ }.

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::close_path.

template <>
path_data_item::rel_curve_to path_data_item::get() const;

template <>
path_data_item::rel_curve_to path_data_item::get(error_code& ec) const noexcept;

Returns: A copy of the stored path_data_item::rel_curve_to object.

If an error occurs and the function was called with an error_code& argument, returns path_data_item::rel_curve_to{ }.

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::rel_curve_to.

template <>
path_data_item::new_sub_path path_data_item::get() const;

template <>
path_data_item::new_sub_path path_data_item::get(error_code& ec) const noexcept;

Returns: A copy of the stored path_data_item::new_sub_path object.

If an error occurs and the function was called with an error_code& argument, returns path_data_item::new_sub_path{ }.

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::new_sub_path.

template <>
path_data_item::line_to path_data_item::get() const;

template <>
path_data_item::line_to path_data_item::get(error_code& ec) const noexcept;
Returns: A copy of the stored path_data_item::line_to object.
If an error occurs and the function was called with an error_code& argument, returns path_data_item::line_to{}. 

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::line_to.

template <>
path_data_item::move_to path_data_item::get() const;
template <>
path_data_item::move_to path_data_item::get(error_code& ec) const noexcept;

Returns: A copy of the stored path_data_item::move_to object.
If an error occurs and the function was called with an error_code& argument, returns path_data_item::move_to{}. 

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::move_to.

template <>
path_data_item::rel_line_to path_data_item::get() const;
template <>
path_data_item::rel_line_to path_data_item::get(error_code& ec) const noexcept;

Returns: A copy of the stored path_data_item::rel_line_to object.
If an error occurs and the function was called with an error_code& argument, returns path_data_item::rel_line_to{}. 

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::rel_line_to.

template <>
path_data_item::rel_move_to path_data_item::get() const;
template <>
path_data_item::rel_move_to path_data_item::get(error_code& ec) const noexcept;

Returns: A copy of the stored path_data_item::rel_move_to object.
If an error occurs and the function was called with an error_code& argument, returns path_data_item::rel_move_to{}. 

Throws: As specified in Error reporting (3).

Error conditions: errc::operation_not_permitted if !this->has_data().
errc::invalid_argument if this->type() != path_data_type::rel_move_to.

10.4 Class path

The path class represents an immutable resource wrapper containing a path geometry graphics resource.

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

A path object shall be usable with any surface or surface-derived object.
10.4.1 path synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 { class path {
    public:
        // 10.4.2, construct/copy/destroy:
        path() = delete;
        explicit path(const path_factory& pb);
        path(const path_factory& pb, error_code& ec) noexcept;
        explicit path(const vector<path_data_item>& p);
        path(const vector<path_data_item>& p, error_code& ec) noexcept;
        path(const path&) noexcept;
        path& operator=(const path&) noexcept;
        path(path&&) noexcept;
        path& operator=(path&&) noexcept;
    };
} } }

10.4.2 path constructors and assignment operators

explicit path(const path_factory& pb);
path(const path_factory& pb, error_code& ec) noexcept;
explicit path(const vector<path_data_item>& p);
path(const vector<path_data_item>& p, error_code& ec) noexcept;

Effects: Constructs an object of class path. Implementations shall create a path geometry graphics resource from the path geometries contained in p or pb.data_ref() as if they followed the procedure set forth in 10.1.2.

Throws: As specified in Error reporting (3).

Remarks: It is unspecified whether a path object shall require further processing when it is passed as an argument to a surface or surface-derived object.

Implementations should avoid or minimize the need for further processing of a path object after it has been constructed.

Error conditions: errc::not_enough_memory if there was a failure to allocate memory.
io2d_error::no_current_point if, when processing the path geometries, an operation was encountered which required a current point and the current path geometry had no current point.
io2d_error::invalid_matrix if, when processing the path geometries, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

10.5 Class path_factory

The path_factory class is a factory class used to create path geometry collection data from which path objects are created.

10.5.1 path_factory synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 { class path_factory {
    // 10.5.2, construct/copy/destroy:
    path_factory() noexcept;
    path_factory(const path_factory&);
    path_factory& operator=(const path_factory&);
    path_factory(path_factory&&) noexcept;
    path_factory& operator=(path_factory&&) noexcept;

§ 10.5.1
// 10.5.3, modifiers:
void append(const path_factory& p);
void append(const path_factory& p, error_code& ec) noexcept;
void append(const vector<path_data_item>& p);
void append(const vector<path_data_item>& p, error_code& ec) noexcept;
void new_sub_path();
void new_sub_path(error_code& ec) noexcept;
void close_path();
void close_path(error_code& ec) noexcept;
void arc(const vector_2d& center, double radius, double angle1,
    double angle2);
void arc(const vector_2d& center, double radius, double angle1,
    double angle2, error_code& ec) noexcept;
void arc_negative(const vector_2d& center, double radius, double angle1,
    double angle2);
void arc_negative(const vector_2d& center, double radius, double angle1,
    double angle2, error_code& ec) noexcept;
void curve_to(const vector_2d& pt0, const vector_2d& pt1,
    const vector_2d& pt2);
void curve_to(const vector_2d& pt0, const vector_2d& pt1,
    const vector_2d& pt2, error_code& ec) noexcept;
void line_to(const vector_2d& pt);  
void line_to(const vector_2d& pt, error_code& ec) noexcept;
void move_to(const vector_2d& pt);  
void move_to(const vector_2d& pt, error_code& ec) noexcept;
void rectangle(const experimental::io2d::rectangle& r);  
void rectangle(const experimental::io2d::rectangle& r,
    error_code& ec) noexcept;
void rel_curve_to(const vector_2d& dpt0, const vector_2d& dpt1,
    const vector_2d& dpt2);
void rel_curve_to(const vector_2d& dpt0, const vector_2d& dpt1,
    const vector_2d& dpt2, error_code& ec) noexcept;
void rel_line_to(const vector_2d& dpt);  
void rel_line_to(const vector_2d& dpt, error_code& ec) noexcept;
void rel_move_to(const vector_2d& dpt);  
void rel_move_to(const vector_2d& dpt, error_code& ec) noexcept;
void transform_matrix(const matrix_2d& m);  
void transform_matrix(const matrix_2d& m, error_code& ec) noexcept;
void origin(const vector_2d& pt);  
void origin(const vector_2d& pt, error_code& ec) noexcept;
void clear() noexcept;

// 10.5.4, observers:
experimental::io2d::rectangle path_extents() const;
experimental::io2d::rectangle path_extents(error_code& ec) 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;
vector<path_data_item> data() const;
vector<path_data_item> data(error_code& ec) const noexcept;
path_data_item data_item(unsigned int index) const;
path_data_item data_item(unsigned int index, error_code& ec) const noexcept;
const vector<path_data_item>& data_ref() const noexcept;

private:
    vector<path_data_item> _Data; // exposition only
    bool _Has_current_point;    // exposition only
    vector_2d _Current_point;   // exposition only
    vector_2d _Last_move_to_point; // exposition only
    matrix_2d _Transform_matrix; // exposition only
    vector_2d _Origin;          // exposition only
};

10.5.2 path_factory constructors and assignment operators [pathfactory.cons]

path_factory();
1
    Effects: Constructs an object of type path_factory.
2    Postconditions: _Data.empty() == true.
3    _Has_current_point == false.
4        _Transform_matrix == matrix_2d::init_identity().
5    _Origin == vector_2d.

10.5.3 path_factory modifiers [pathfactory.modifiers]

void append(const path_factory& p);
void append(const path_factory& p, error_code& ec) noexcept;
1
    Postconditions: This section is forthcoming in a future revision.

void append(const vector<path_data_item>& p);
void append(const vector<path_data_item>& p, error_code& ec) noexcept;
2
    Postconditions: This section is forthcoming in a future revision.

void new_sub_path();
void new_sub_path(error_code& ec) noexcept;
3
    Effects: _Data.emplace_back(path_data_item::new_sub_path()).
4    _Has_current_point = false.
5    Throws: As specified in Error reporting (3).
6    Remarks: In the event of an error, the object shall not be modified.
7    Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void close_path();
void close_path(error_code& ec) noexcept;
8
    Effects: If _Has_current_point == true:
9
(8.1)    _Data.emplace_back(path_data_item::close_path()).
(8.2)    _Current_point = _Last_move_to_point.

9    Throws: As specified in Error reporting (3).
10    Remarks: In the event of an error, the object shall not be modified.
11    Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.
void arc(const vector_2d& center, double radius, double angle1, double angle2);
void arc(const vector_2d& center, double radius, double angle1, double angle2, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::arc(center, radius, angle1, angle2)).
_Current_point == vector_2{ radius * cos(angle2), -(radius * -sin(angle2)) } + center.
If _Has_current_point == false:
   — _Last_move_to_point == vector_2{ radius * cos(angle1), -(radius * -sin(angle1)) } + center.
(14.1)
   — _Has_current_point == true.
(14.2)

Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void arc_negative(const vector_2d& center, double radius, double angle1, double angle2);
void arc_negative(const vector_2d& center, double radius, double angle1, double angle2, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::arc_negative(center, radius, angle1, angle2)).
_Current_point == vector_2{ radius * cos(angle1), radius * -sin(angle1) } + center.
If _Has_current_point == false:
   — _Last_move_to_point == vector_2{ radius * cos(angle2), radius * -sin(angle2) } + center.
(20.1)
   — _Has_current_point == true.
(20.2)

Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void curve_to(const vector_2d& pt0, const vector_2d& pt1, const vector_2d& pt2);
void curve_to(const vector_2d& pt0, const vector_2d& pt1, const vector_2d& pt2, error_code& ec) noexcept;

Effects: If _Has_current_point == false:
(24.1)
   — _Data.reserve(_Data.size() + 2U).
(24.2)
   — *this.move_to(pt0).
   _Data.emplace_back(path_data_item::curve_to(pt0, pt1, pt2)).

Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void line_to(const vector_2d& pt);
void line_to(const vector_2d& pt, error_code& ec) noexcept;
Effects: _Data.emplace_back(path_data_item::line_to(pt)).

If _Has_current_point == false:

(30.1) _Last_move_to_point = pt.
(30.2) _Has_current_point = true.

_Current_point = pt.

Throws: As specified in Error reporting (3).

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void move_to(const vector_2d& pt);
void move_to(const vector_2d& pt, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::move_to(pt)).

_Has_current_point = true.
_Current_point = pt.

Throws: As specified in Error reporting (3).

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void rectangle(const experimental::io2d::rectangle& r);
void rectangle(const experimental::io2d::rectangle& r,
  error_code& ec) noexcept;

Effects:

1. _Data.reserve(_Data.size() + 5U).
2. *this.move_to({ r.x(), r.y() }).
3. *this.rel_line_to({ r.width(), 0.0 }).
4. *this.rel_line_to({ 0.0, r.height() }).
5. *this.rel_line_to({ -r.width(), 0.0 }).
6. *this.close_path().

Throws: As specified in Error reporting (3).

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void rel_curve_to(const vector_2d& dpt0, const vector_2d& dpt1,
  const vector_2d& dpt2);
void rel_curve_to(const vector_2d& dpt0, const vector_2d& dpt1,
  const vector_2d& dpt2, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::rel_curve_to(dpt0, dpt1, dpt2)).

_Current_point = dpt2 + _Current_point.

Throws: As specified in Error reporting (3).

Remarks: In the event of an error, the object shall not be modified.

Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

io2d_error::no_current_point if _Has_current_point == false.
void rel_line_to(const vector_2d& dpt);
void rel_line_to(const vector_2d& dpt, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::rel_line_to(pt)).
_Current_point = dpt + _Current_point.

Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.
io2d_error::no_current_point if _Has_current_point == false.

void rel_move_to(const vector_2d& dpt);
void rel_move_to(const vector_2d& dpt, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::rel_move_to(dpt)).
_Current_point = dpt + _Current_point.

Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.
io2d_error::no_current_point if _Has_current_point == false.

void transform_matrix(const matrix_2d& m);
void transform_matrix(const matrix_2d& m, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::change_matrix(m)).
_Transform_matrix = m.

Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void origin(const vector_2d& pt);
void origin(const vector_2d& pt, error_code& ec) noexcept;

Effects: _Data.emplace_back(path_data_item::change_origin(pt)).
_Origin = pt.
Throws: As specified in Error reporting (3).
Remarks: In the event of an error, the object shall not be modified.
Error conditions: errc::not_enough_memory if the attempt to add the path_data_item failed.

void clear() noexcept;

Postconditions: _Data.empty() == true.
_Has_current_point == false.
_Transform_matrix == matrix_2d::init_identity{ }.
_Origin == vector_2d{ }.

§ 10.5.3
10.5.4  path_factory observers

experimental::io2d::rectangle path_extents() const;
experimental::io2d::rectangle path_extents(error_code& ec) const noexcept;

Returns: A rectangle object which contains the extents of the path segments, including degenerate path segments, in _Data when it is processed as described in 10.1.2. [ 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 ]

Throws: As specified in Error reporting (3).

Error conditions: io2d_error::invalid_matrix if _Data includes a change_matrix operation which establishes a non-invertible matrix_2d as the transformation matrix and that matrix must subsequently be inverted in order to process the path geometries.

bool has_current_point() const noexcept;

Returns: _Has_current_point.

vector_2d current_point() const;
vector_2d current_point(error_code& ec) const noexcept;

Returns: _Current_point.

Throws: As specified in Error reporting (3).

Error conditions: io2d_error::no_current_point if _Has_current_point == false.

matrix_2d transform_matrix() const noexcept;

Returns: _Transform_matrix.

vector_2d origin() const noexcept;

Returns: _Origin.

vector<path_data_item> data() const;
vector<path_data_item> data(error_code& ec) const noexcept;

Returns: A copy of _Data.

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if there was a failure to allocate memory.

path_data_item data_item(unsigned int index) const;
path_data_item data_item(unsigned int index, error_code& ec) const noexcept;

Returns: _Data.at(index).

Throws: As specified in Error reporting (3).

Error conditions: io2d_error::invalid_index if _Data.size() <= index.

const vector<path_data_item>& data_ref() const noexcept;

Returns: _Data.
11 Fonts

Fonts describe how text should be rendered and composed. This Technical Specification leaves much of how the rendering and composing happens up to the implementors.

Fonts exist to describe how text is rendered. The only requirements that exist in this Technical Specification are:

1. The text to be rendered is in the UTF-8 character encoding.
2. The default font face of the implementation shall correctly render all of the characters of the Unicode Basic Multilingual Plane C0 Controls and Basic Latin.

11.1 Enum class font_slant

11.1.1 font_slant Summary

The `font_slant` enum class specifies the slant requested for rendering text.

These values have different meanings for different scripts. For some scripts they may have no meaning at all. Further, not all typefaces will support every value.

As such, these values are requests which implementations should honor if possible.

See Table 4 for the meaning of each `font_slant` enumerator.

11.1.2 font_slant Synopsis

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

11.1.3 font_slant Enumerators

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>The text shall be rendered in whatever is a normal type for the font. If a font has both an italic type and an oblique type but does not have another type, then the italic type shall be the normal type for the font unless the font includes data that specifies otherwise. [Note: If a font only has an italic type or only an oblique type then that is the normal type for the font. — end note]</td>
</tr>
<tr>
<td>italic</td>
<td>The text should be rendered in whatever is an italic type for the font. If a font does not have an italic type but does have an oblique type, the oblique type shall be used if <code>italic</code> is requested. If a font has neither an italic type nor and oblique type, the normal type shall be used.</td>
</tr>
</tbody>
</table>
11.2 Enum class font_weight

11.2.1 font_weight Synopsis

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

11.2.2 font_weight Summary

1 The font_weight enum class specifies the font weight for rendering text.
2 See Table 5 for the meaning of each enumerator.

11.2.3 font_weight Enumerators

```
Table 5 — font_weight enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>The text shall be rendered in whatever is a normal weight for the script.</td>
</tr>
<tr>
<td>bold</td>
<td>The text shall be rendered in whatever is a bold weight for the script.</td>
</tr>
</tbody>
</table>
```

11.3 Enum class subpixel_order

11.3.1 subpixel_order Summary

1 The subpixel_order enum class is used to request a specific order of color channels for each pixel of an output device. When a surface object’s font_options object has its antialias value set to antialias::subpixel and its subpixel_order value set to one of these values, an implementation should use the specified subpixel_order to render text. See Table 6 for the meaning of each subpixel_order enumerator.

11.3.2 subpixel_order Synopsis

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

Table 4 — font_slant enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>oblique</td>
<td>The text should be rendered in whatever is an oblique type for the font. If a font does not have an oblique type but does have an italic type, the italic type shall be used if oblique is requested. If a font has neither an italic type nor and oblique type, the normal type shall be used.</td>
</tr>
</tbody>
</table>
11.3.3 subpixel_order Enumerators

Table 6 — subpixel_order enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_subpixel_order</td>
<td>The implementation should use the target surface object’s default subpixel order.</td>
</tr>
<tr>
<td>horizontal_rgb</td>
<td>The color channels should be arranged horizontally starting with red on the left, followed by green, then blue.</td>
</tr>
<tr>
<td>horizontal_bgr</td>
<td>The color channels should be arranged horizontally starting with blue on the left, followed by green, then red.</td>
</tr>
<tr>
<td>vertical_rgb</td>
<td>The color channels should be arranged vertically starting with red on the top, followed by green, then blue.</td>
</tr>
<tr>
<td>vertical_bgr</td>
<td>The color channels should be arranged vertically starting with blue on the top, followed by green, then red.</td>
</tr>
</tbody>
</table>

11.4 Class font_options

11.4.1 font_options synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class font_options {
    public:
        // 11.4.3, construct/copy/destroy:
        font_options() noexcept;
        font_options(const font_options& other) noexcept;
        font_options& operator=(const font_options& other) noexcept;
        font_options(font_options&& other) noexcept;
        font_options& operator=(font_options&& other) noexcept;
        font_options(std::experimental::io2d::antialias a,
                     std::experimental::io2d::subpixel_order so) noexcept;

        // 11.4.4, modifiers:
        void antialias(std::experimental::io2d::antialias value) noexcept;
        void subpixel_order(std::experimental::io2d::subpixel_order value) noexcept;

        // 11.4.5, observers:
        std::experimental::io2d::antialias antialias() const noexcept;
        std::experimental::io2d::subpixel_order subpixel_order() const noexcept;
    private:
        std::experimental::io2d::antialias _Antialias; // exposition only
        std::experimental::io2d::subpixel_order _Subpixel_order; // exposition only
    };
}}}}

11.4.2 font_options Description

The font_options class describes an object that holds values which specify certain aspects of how text should be rendered.

11.4.3 font_options constructors and assignment operators
font_options() noexcept;

Effects: Constructs an object of type font_options.

Postconditions: _Antialias == std::experimental::io2d::antialias::default_antialias.
_subpixel_order == std::experimental::io2d::subpixel_order::default_subpixel_order.

font_options(std::experimental::io2d::antialias a,
std::experimental::io2d::subpixel_order so) noexcept;

Effects: Constructs an object of type font_options.

Postconditions: _Antialias == a.
_subpixel_order == so.

11.4.4 font_options modifiers

void antialias(std::experimental::io2d::antialias value) noexcept;

Postconditions: _Antialias == value.

void subpixel_order(std::experimental::io2d::subpixel_order value) noexcept;

Postconditions: _Subpixel_order == value.

11.4.5 font_options observers

std::experimental::io2d::antialias antialias() const noexcept;

Returns: _Antialias.

std::experimental::io2d::subpixel_order subpixel_order() const noexcept;

Returns: _Subpixel_order.

11.5 Class font_face

11.5.1 font_face synopsis

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

class font_face {

public:

// See 2.3
typedef implementation-defined native_handle_type;
native_handle_type native_handle() const noexcept;

// 11.5.3, construct/copy/destroy:
font_face() = delete;
font_face(const font_face&) noexcept;
font_face& operator=(const font_face&) noexcept;
font_face(font_face&& other) noexcept;
font_face& operator=(font_face&& other) noexcept;
virtual ~font_face();
};
} } } }

11.5.2 font_face Description

The font_face class is an opaque resource wrapper. It represents a font that is used to render text. It shall not be directly instantiable.
11.5.3  font_face constructors and assignment operators

virtual ~font_face();

Effects: Destroys an object of class font_face.

11.6  Class simple_font_face

11.6.1  simple_font_face synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    class simple_font_face : public font_face {
    public:
        // See 2.3
        typedef implementation-defined native_handle_type; // exposition only
        native_handle_type native_handle() const noexcept; // exposition only

        // 11.6.3, construct/copy/destroy:
        simple_font_face() = delete;
        simple_font_face(const simple_font_face&) noexcept;
        simple_font_face& operator=(const simple_font_face&) noexcept;
        simple_font_face(simple_font_face&& other) noexcept;
        simple_font_face& operator=(simple_font_face&& other) noexcept;
        simple_font_face(const string& typeface,
                         std::experimental::io2d::font_slant fs,
                         std::experimental::io2d::font_weight fw);
        simple_font_face(const string& typeface,
                         std::experimental::io2d::font_slant fs,
                         std::experimental::io2d::font_weight fw, error_code& ec) noexcept;

        // 11.6.4, observers:
        string typeface() const;
        void typeface(string& str, error_code& ec) const noexcept;
        ::std::experimental::io2d::font_slant font_slant() const noexcept;
        ::std::experimental::io2d::font_weight font_weight() const noexcept;

    private:
        string _Typeface; // exposition only
        std::experimental::io2d::font_slant _Font_slant; // exposition only
        std::experimental::io2d::font_weight _Font_weight; // exposition only
    };
}}}

11.6.2  simple_font_face Description

The simple_font_face class represents a font that is used to render text. It only guarantees rudimentary
text rendering capabilities.

It shall correctly render text written horizontally in left-to-right scripts where there is a one-to-one mapping
between characters and glyphs.

It may correctly render:

(3.1)  — right-to-left scripts;
(3.2)  — bi-directional text;
(3.3)  — vertically-oriented text;
(3.4)  — combining character sequences.

4 It may correctly handle some or all aspects of complex text rendering, e.g.:

(4.1)  — bi-directional text;
(4.2)  — context sensitive shaping;
(4.3)  — ligatures;
(4.4)  — cursive scripts;
(4.5)  — text where the proper display order (the order text is rendered) differs from the logical order (the order text is typed on a keyboard).

11.6.3  

§ 11.6.3 simple_font_face constructors and assignment operators [simplefontface.cons]

```cpp
simple_font_face(const string& typeface,
    std::experimental::io2d::font_slant fs,
    std::experimental::io2d::font_weight fw);

simple_font_face(const string& typeface,
    std::experimental::io2d::font_slant fs,
    std::experimental::io2d::font_weight fw, error_code& ec) noexcept;
```

1 Effects: Constructs an object of class simple_font_face. A font that can be used to render text to a surface object is associated with the object.

2 Implementations should perform all possible actions that are required to use the font to render text rather than delay such actions until the font is actually used.

3 Postconditions: _Typeface == typeface if the implementation located a usable typeface by that name, otherwise _Typeface will be set to the name of the implementation-defined nearest match typeface which the implementation selected instead. [Note: Implementations have been given control over deciding what the 'nearest match' is because typeface names are arbitrary and any individual with the proper tools can make a new typeface.

Ideally an implementation which could not find an exact match would search for nearest matches in the following order. First it would run a search for typefaces with names that closely matched the requested typeface string based on common misspellings and typos. If no match was found it would then search some form of lookup table which stored distinguishing characteristics of a large number of typefaces (e.g. those which are provided by operating systems and applications which have an install base in excess of one million copies). If it found a listing in the table for the requested typeface or, barring that, a typeface which was a near match due to misspellings and typos, it would then use the characteristics provided by the table to make the nearest possible match. Optimally the table would include first preference matches where one font is known to have been developed to match another font as closely as legally permissible. The implementation would then use the results of the lookup table to find the closest available matching typeface. Lastly, only if the previous methods had failed would the implementation provide a font of its choosing.

Nonetheless, implementations may, with equal validity, return as being the nearest match the same typeface regardless of any information that could be gleaned from the unmatched request and without any consideration of spelling mistakes or typos. — end note]

4 _Font_slant_ shall equal the value determined according to the requirements set forth in 11.1 after the value of _Typeface_ has been determined. If that font_slant value is different than fs, _Font_slant_ shall be set to the actual value used. [Example: If font_slant::italic is requested but an italic type does not exist for the chosen typeface, then if an oblique type exists, it is chosen and
_Font_slant == font_slant::oblique, otherwise the normal type is chosen and _Font_slant == font_slant::normal. — end example]

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if there was a failure to allocate memory.

### 11.6.4 simple_font_face observers

```cpp
string typeface() const;
Returns: _Typeface.
```

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if there was a failure to allocate memory.

```cpp
void typeface(string& str, error_code& ec) const noexcept;
Effects: str = _Typeface.
```

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if there was a failure to allocate memory.

```cpp
std::experimental::io2d::font_slant font_slant() const noexcept;
Returns: _Font_slant.
```

### 11.7 Class font_extents

#### 11.7.1 font_extents synopsis

```cpp
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class font_extents {
public:
  // 11.7.3, construct/copy/move/destroy:
  font_extents() noexcept;
  font_extents(const font_extents& other) noexcept;
  font_extents& operator=(const font_extents& other) noexcept;
  font_extents(font_extents&& other) noexcept;
  font_extents& operator=(font_extents&& other) noexcept;
  font_extents(double ascent, double descent, double height) noexcept;

  // 11.7.4, modifiers:
  void ascent(double value) noexcept;
  void descent(double value) noexcept;
  void height(double value) noexcept;

  // 11.7.5, observers:
  double ascent() const noexcept;
  double descent() const noexcept;
  double height() const noexcept;

private:
  double _Asc;  // exposition only
```

§ 11.7.1
11.7.2 font_extents Description

The class font_extents describes metric information for a font.

It is used by a surface object to report certain metrics of its currently selected font in the surface object’s untransformed coordinate space units.

These metrics cover all glyphs in a font and thus may be noticeably larger than the values obtained by getting the text_extents for a particular string.

[Note: This object’s observable values can be manipulated by library users for their convenience. But since the font_extents object returned by surface::font_extents() is not a reference or a pointer, the changes do not reflect back to the surface or its current font. — end note]

11.7.3 font_extents constructors and assignment operators

font_extents() noexcept;

Effects: Constructs an object of type font_extents.

Postconditions: _Asc == 0.0.
   _Desc == 0.0.
   _Height == 0.0.

font_extents(double ascent, double descent, double height) noexcept;

Effects: Constructs an object of type font_extents.

Postconditions: _Asc == ascent.
   _Desc == descent.
   _Height == height.

11.7.4 font_extents modifiers

void ascent(double value) noexcept;

Postconditions: _Asc == value.

void descent(double value) noexcept;

Postconditions: _Desc == descent.

void height(double value) noexcept;

Postconditions: _Height == value.

11.7.5 font_extents observers

double ascent() const noexcept;

Returns: _Asc.

Remarks: This value is the distance in untransformed coordinate space units from the top of the font’s bounding box to the font’s baseline.

Some glyphs may extend slightly above the top of the font’s bounding box due to hinting or for aesthetic reasons.
double descent() const noexcept;

    Returns: _Desc.

    Remarks: This value is the distance in untransformed coordinate space units from the bottom of the
    font’s bounding box to the font’s baseline.

    Some glyphs may extend slightly below the bottom of the font’s bounding box due to hinting or for
    aesthetic reasons.

    [ Note: Some font rendering technologies express this value as a negative value. Because it is defined
    here as a distance from the baseline, the value should typically be positive or zero. It would only
    be negative if the font’s baseline was set below the bottom of its bounding box, which, while highly
    unlikely, is not impossible. — end note ]

double height() const noexcept;

    Returns: _Height.

    Remarks: This value is the font designer’s suggested distance, in untransformed coordinate space units,
    from the baseline of one line of text to the baseline of a consecutive line of text.

    This value is may be greater than the sum of ascent() and descent(). This occurs when a font
    includes a value known as a line gap or as external leading, which is additional whitespace added for
    aesthetic reasons.

    Fonts whose height() is equal to their ascent() + descent() likely include line gap in their ascent
    or descent rather than specifying it separately.

11.8 Class text_extents

11.8.1 text_extents synopsis

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

class text_extents {

  // 11.8.3, construct/copy/move/destroy:
  text_extents() noexcept;
  text_extents(const text_extents& other) noexcept;
  text_extents& operator=(const text_extents& other) noexcept;
  text_extents(font_extents&& other) noexcept;
  text_extents& operator=(font_extents&& other) noexcept;
  text_extents(double xBearing, double yBearing, double width,
               double height, double xAdvance, double yAdvance) noexcept;

  // 11.8.4, modifiers:
  void x_bearing(double value) noexcept;
  void y_bearing(double value) noexcept;
  void width(double value) noexcept;
  void height(double value) noexcept;
  void x_advance(double value) noexcept;
  void y_advance(double value) noexcept;

  // 11.8.5, observers:
  double x_bearing() const noexcept;
  double y_bearing() const noexcept;
  double width() const noexcept;
  double height() const noexcept;
  double x_advance() const noexcept;
  double y_advance() const noexcept;

§ 11.8.1
private:
    double _X_bear; // exposition only
    double _Y_bear; // exposition only
    double _Width; // exposition only
    double _Height; // exposition only
    double _X_adv; // exposition only
    double _Y_adv; // exposition only
};
} } } }

11.8.2 text_extents Description

1 The class text_extents describes extents for a string.

2 It is used by a surface object to report the extents of a string in the surface object’s untransformed coordinate space units if the string were rendered with the currently selected font.

3 [Note: This object’s observable values can be manipulated by library users for their convenience. But since the text_extents object returned by surface::text_extents() is not a reference or a pointer, the changes do not reflect back to the surface or its current font. — end note]

11.8.3 text_extents constructors and assignment operators

    text_extents() noexcept;
1    Effects: Constructs an object of type text_extents.
2    Postconditions: _X_bear == 0.0.
3        _Y_bear == 0.0.
4        _Width == 0.0.
5        _Height == 0.0.
6        _X_adv == 0.0.
7        _Y_adv == 0.0.

    text_extents(double xBearing, double yBearing, double width,
8      double height, double xAdvance, double yAdvance) noexcept;
9    Effects: Constructs an object of type text_extents.
10    Postconditions: _X_bear == xBearing.
11        _Y_bear == yBearing.
12        _Width == width.
13        _Height == height.
14        _X_adv == xAdvance.
15        _Y_adv == yAdvance.

11.8.4 text_extents modifiers

    void x_bearing(double value) noexcept;
1    Postconditions: _X_bear == value.

    void y_bearing(double value) noexcept;
2    Postconditions: _Y_bear == value.
void width(double value) noexcept;

Postconditions: _Width == value.

void height(double value) noexcept;

Postconditions: _Height == value.

void x_advance(double value) noexcept;

Postconditions: _X_adv == value.

void y_advance(double value) noexcept;

Postconditions: _Y_adv == value.

11.8.5 text_extents observers

double x_bearing() const noexcept;

Returns: _X_bear.

Remarks: This value is the x axis offset of the leftmost visible part of the text as rendered from the x coordinate of the specified position at which to draw the text.

Leading and trailing spaces can affect this value due to the fact that spaces change the position of other text and thus can change the position of the first visible text that is rendered.

[Note: Because this value is an offset from the specified position and is given in untransformed units, it remains the same regardless of the position at which the text will be drawn.

This value will typically be negative, zero, or slightly positive depending on the font used and the text being rendered (e.g. scripts that are written right-to-left will normally have a negative x_bearing() value). — end note]

double y_bearing() const noexcept;

Returns: _Y_bear.

Remarks: This value is the y axis offset of the topmost visible part of the text as rendered from the y coordinate of the specified position at which to draw the text.

[Note: This value may range from negative to positive depending on the font origin chosen by the font designer. Usually this value is negative. — end note]

double width() const noexcept;

Returns: _Width.

Remarks: This value is the width of the text as rendered from its leftmost visible part to its rightmost visible part.

This value may include a de minimus amount of whitespace, e.g. 1 to 2 pixels when pixels are the coordinate space unit. This allowance is meant to cover discrepancies between expected rendering results and actual results which can arise due to techniques such as font hinting, antialiasing, and subpixel rendering.

double height() const noexcept;
Returns: _Height.

Remarks: This value is the height of the text as rendered from its topmost visible part to its bottommost visible part.

This value may include a de minimus amount of whitespace, e.g. 1 to 2 pixels when pixels are the coordinate space unit. This allowance is meant to cover discrepancies between expected rendering results and actual results which can arise due to techniques such as font hinting, antialiasing, and subpixel rendering.

double x_advance() const noexcept;

Returns: _X_adv.

Remarks: This value is amount to add to the x coordinate of the original specified position in order to properly draw text that will immediately follow this text on the same line.

In vertically oriented text, this value will typically be zero.

double y_advance() const noexcept;

Returns: _Y_adv.

Remarks: This value is amount to add to the y coordinate of the original specified position in order to properly draw text that will immediately follow this text on the same line.

In horizontally oriented text, this value will typically be zero.
12 Brushes

1 Brushes serve as sources of visual data for composing operations.

2 A brush has its own mutable coordinate space (by default the standard coordinate space). It also possesses a mutable extend value and a mutable filter value.

3 A brush is created using one of four factory classes:

(3.1) — solid_color_brush_factory,

(3.2) — linear_brush_factory,

(3.3) — radial_brush_factory, and

(3.4) — surface_brush_factory.

4 Excluding brushes created using a solid_color_brush_factory object, brushes produce very different composing operation results depending on the values of their mutable state.

5 Brushes created using either a linear_brush_factory object or a radial_brush_factory object are considered to be gradient brushes. They share similarities with each other that are not shared by brushes created using the other brush factories. This is discussed in more detail below (12.1).

12.1 Gradient brushes

12.1.1 Color stops

1 A gradient has as part of its observable state a collection of color stops.

2 The collection of color stops has an implementation-defined maximum size.

3 Associated with the collection of color stops is a type called the size_type, which is an implementation-defined unsigned integer type that is large enough that it can serve as an index to address each color stop in any collection of color stops as if the collection of color stops was accessed as an array of color stops.

4 The size of a collection of color stops is the number of color stops it currently contains; its type is size_type.

5 When a color stop is added to a collection of color stops it is assigned an index value that is equal to the size of the collection of color stops before the color stop is added.

6 A valid index value for a collection of color stops is an unsigned integer value in the range \( (0, \text{size}) \).

7 Color stops in a collection of color stops can be replaced.

8 A color stop which replaces another color stop in a collection of color stops is assigned the index value of the color stop it replaced. [Example: If 3 color stops are added to a collection of color stops and then the color stop at index 1 is replaced, the next color stop which is added will be assigned the index value 3. — end example]

9 Color stops in a collection of color stops can be removed.

10 When a color stop with the valid index value \( i \) is removed from a collection of color stops, the index value of every color stop in the collection of color stops which had an index value greater than \( i \) has its index value decremented by 1. [Note: The size of the collection of color stops is reduced by 1 as the result of the removal of one of its color stops. — end note]
**12.1.2 Linear gradients**

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 12.1.2 is inapplicable to degenerate linear gradients.

5. The begin point and end point of a linear gradient define a line segment, with a color stop offset value of 0.0 corresponding to the begin point and a color stop offset value of 1.0 corresponding to the end point.

6. Color stop offset values in the range (0, 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 $dp ÷ dt$.

   d) The offset value shall be negative if

   - $p$ 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,

   - 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.

**12.1.3 Radial gradients**

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:

   - its start circle has a negative radius; or,

   - its end circle has a negative radius; or,

   - 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,
its start circle has a radius of 0.0 and its end circle has a radius of 0.0.

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 12.1.3 is inapplicable to degenerate radial gradients.

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.

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.

A radial gradient shall be rendered as a continuous series of interpolated circles defined by the following equations:

\[
\begin{align*}
    x(o) &= x_{\text{start}} + o \times (x_{\text{end}} - x_{\text{start}}) \\
    y(o) &= y_{\text{start}} + o \times (y_{\text{end}} - y_{\text{start}}) \\
    \text{radius}(o) &= \text{radius}_{\text{start}} + o \times (\text{radius}_{\text{end}} - \text{radius}_{\text{start}})
\end{align*}
\]

where \( o \) is a color stop offset value.

The range of potential values for \( o \) shall be determined by the extend value of the brush object created using the radial gradient:

- For extend::none, the range of potential values for \( o \) is \([0, 1]\).
- For all other extend values, the range of potential values for \( o \) is \([\text{numeric_limits<double>::lowest()}, \text{numeric_limits<double>::max()}]\).

The interpolated circles shall be rendered starting from the smallest potential value of \( o \).

An interpolated circle shall not be rendered if its value for \( o \) results in \( \text{radius}(o) \) evaluating to a negative value.

12.1.4 Sampling from gradients

For any offset value \( o \), its color value shall be determined according to the following rules:

- 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 12.1.4 is inapplicable.
- 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 12.1.4 is inapplicable.
- 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 12.1.4 is inapplicable.
- 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 12.1.4 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]
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 §12.1.4 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]

Each color stop has, at most, two adjacent color stops: one to its left and one to its right.

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.

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]

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]

A color stop which has an offset value of 0.0 does not have an adjacent color stop to its left.

A color stop which has an offset value of 1.0 does not have an adjacent color stop to its right.

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.

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.

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.

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.

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.

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.

Where the value of \( o \) is outside the range \([0, 1]\), its color value depends on the extend value of the brush which is created using the gradient.
— If the \texttt{extend} value is \texttt{extend::none}, the color value of \( o \) shall be \texttt{rgba\_color::transparent\_black}().

— If the \texttt{extend} value is \texttt{extend::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.

— If the \texttt{extend} value is \texttt{extend::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. \[ \text{Example: Given } o == 2.1, \text{ after application of this rule } o == 0.1 \text{ and the color value of } o \text{ shall be the same value as if the initial value of } o \text{ was 0.1.} \]

\hspace{1cm} 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. \[ \text{—end example} \]

— If the \texttt{extend} value is \texttt{extend::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. \[ \text{Example: Given } o == 2.8, \text{ after application of this rule } o == 0.8 \text{ and the color value of } o \text{ shall be the same value as if the initial value of } o \text{ was 0.8.} \]

\hspace{1cm} 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.

\hspace{1cm} 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.

\hspace{1cm} 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. \[ \text{—end example} \]

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.

### 12.2 Enum class \texttt{extend}

#### 12.2.1 \texttt{extend} Summary

The \texttt{extend} enum class describes how a pixel’s color should be determined if it is outside the boundary of the source \texttt{brush} or \texttt{surface} during a call to \texttt{surface::fill}, \texttt{surface::mask}, \texttt{surface::paint}, or \texttt{surface::stroke}.

2 See Table 7 for the meaning of each \texttt{extend} enumerator.

#### 12.2.2 \texttt{extend} Synopsis

```cpp
namespace std { namespace experimental { namespace io2d { inline namespace v1 { enum class extend {
    none,
    repeat,
    reflect,
    pad
};
} } } }
```

§ 12.2.2
12.2.3  extend Enumerators

Table 7 — extend enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Any pixel outside of the bounds of the brush or surface which is required to be set shall be set to transparent black.</td>
</tr>
<tr>
<td>repeat</td>
<td>A pixel outside of the bounds of the brush or surface which is required to be set shall be set to the color it would have been set to if the brush or surface was infinitely large and repeated itself in a left-to-right-left-to-right and top-to-bottom-top-to-bottom fashion.</td>
</tr>
<tr>
<td>reflect</td>
<td>A pixel outside of the bounds of the brush or surface which is required to be set shall be set to the color it would have been set to if the brush or surface was infinitely large and repeated itself in a left-to-right-to-left-to-right and top-to-bottom-to-top-to-bottom fashion.</td>
</tr>
<tr>
<td>pad</td>
<td>A pixel outside of the bounds of the brush or surface which is required to be set shall be set to the color of the nearest pixel that is in bounds.</td>
</tr>
</tbody>
</table>

12.3  Enum class filter

12.3.1  filter Summary

The filter enum class specifies the type of filter to use when sampling from a pixmap.

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.

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.

See Table 8 for the meaning of each filter enumerator.

12.3.2  filter Synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
    enum class filter {
        fast,
        good,
        best,
        nearest,
        bilinear
    };
    } } } }

12.3.3  filter Enumerators
Table 8 — filter enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
<td>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]</td>
</tr>
<tr>
<td>good</td>
<td>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]</td>
</tr>
<tr>
<td>best</td>
<td>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]</td>
</tr>
<tr>
<td>nearest</td>
<td>Nearest-neighbor interpolation filtering shall be used.</td>
</tr>
<tr>
<td>bilinear</td>
<td>Bilinear interpolation filtering shall be used.</td>
</tr>
</tbody>
</table>

12.4 Enum class brush_type

12.4.1 brush_type Summary

The brush_type enum class denotes which brush factory was used to form a brush object.

12.4.2 brush_type Synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class brush_type {
    solid_color, surface, linear, radial, mesh
  };
} } } }

12.4.3 brush_type Enumerators

Table 9 — brush_type enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>solid_color</td>
<td>The brush object was created from a solid_color_brush_factory object.</td>
</tr>
<tr>
<td>surface</td>
<td>The brush object was created from a surface_brush_factory object.</td>
</tr>
</tbody>
</table>

§ 12.4.3
### Table 9 — `brush_type` enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>The brush object was created from a <code>linear_brush_factory</code> object.</td>
</tr>
<tr>
<td>radial</td>
<td>The brush object was created from a <code>radial_brush_factory</code> object.</td>
</tr>
<tr>
<td>mesh</td>
<td>The brush object was created from a <code>mesh_brush_factory</code> object.</td>
</tr>
</tbody>
</table>

### 12.5 Class `brush` [brush]

#### 12.5.1 `brush` synopsis [brush.synopsis]

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

    class brush {
        public:
            // 12.5.3, construct/copy/move/destroy:
            brush() = delete;
            brush(const brush&) noexcept;
            brush& operator=(const brush&) noexcept;
            brush(brush&& other) noexcept;
            brush& operator=(brush&& other) noexcept;
            brush(const solid_color_brush_factory& f);
            brush(const solid_color_brush_factory& f, error_code& ec) noexcept;
            brush(const linear_brush_factory& f);
            brush(const linear_brush_factory& f, error_code& ec) noexcept;
            brush(const radial_brush_factory& f);
            brush(const radial_brush_factory& f, error_code& ec) noexcept;
            brush(surface_brush_factory& f);
            brush(surface_brush_factory& f, error_code& ec) noexcept;

            // 12.5.4, modifiers:
            void extend(experimental::io2d::extend e) noexcept;
            void filter(experimental::io2d::filter f) noexcept;
            void matrix(const matrix_2d& m) noexcept;

            // 12.5.5, observers:
            experimental::io2d::extend extend() const noexcept;
            experimental::io2d::filter filter() const noexcept;
            matrix_2d matrix() const noexcept;
            brush_type type() const noexcept;

        private:
            experimental::io2d::extend _Extend; // exposition only
            experimental::io2d::filter _Filter; // exposition only
            matrix_2d _Matrix; // exposition only
            brush_type _Brush_type; // exposition only
    };
}}}}
```

#### 12.5.2 `brush` Description [brush.intro]

1. The class `brush` describes an opaque wrapper for a raster graphics data graphics resource.
2. A `brush` object shall be usable with any `surface` or `surface`-derived object.
A brush object’s graphics data is immutable.

A brush object also has mutable `extend`, `filter`, and `matrix_2d` values which affect how the color and alpha data is used and interpreted.

A brush object has immutable `brush_type` data which indicates the type of brush factory that was used in creating the brush object (See Table 9).

The brush object’s graphics data may have less precision than the graphics data of the brush factory object from which it was created.

[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]

### 12.5.3 brush constructors and assignment operators

```cpp
brush(const solid_color_brush_factory& f);
brush(const solid_color_brush_factory& f, error_code& ec) noexcept;
```

**Effects:** Constructs an object of type brush.

**Postconditions:**

- `_Brush_type` == `brush_type::solid_color`.
- `_Extend` == `std::experimental::io2d::extend::none`.
- `_Filter` == `std::experimental::io2d::filter::fast`.
- `_Matrix` == `matrix_2d::init_identity()`.

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

**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.

```cpp
brush(const linear_brush_factory& f);
brush(const linear_brush_factory& f, error_code& ec) noexcept;
```

**Effects:** Constructs an object of type brush.

**Postconditions:**

- `_Brush_type` == `brush_type::linear`.
- `_Extend` == `std::experimental::io2d::extend::none`.
- `_Filter` == `std::experimental::io2d::filter::good`.
- `_Extend` == `std::experimental::io2d::extend::none`.

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

**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.

```cpp
brush(const radial_brush_factory& f);
brush(const radial_brush_factory& f, error_code& ec) noexcept;
```

**Effects:** Constructs an object of type brush.

**Postconditions:**

- `_Brush_type` == `brush_type::radial`.
- `_Extend` == `std::experimental::io2d::extend::none`.

§ 12.5.3
_Filter == std::experimental::io2d::extend::good.
_Extend == std::experimental::io2d::extend::none.

 Throws: As specified in Error reporting (3).

 Error conditions: errc::not_enough_memory if there was a failure to allocate memory.
o2d_error::invalid_status if there was a failure to allocate a resource other than memory.

brush(surface_brush_factory& f);
brush(surface_brush_factory& f, error_code& ec) noexcept;

 Effects: Constructs an object of type brush.
 Postconditions: _Brush_type == brush_type::surface.
 _Extend == std::experimental::io2d::extend::none.
 _Filter == std::experimental::io2d::extend::good.
 _Extend == std::experimental::io2d::extend::none.

 Throws: As specified in Error reporting (3).

 Error conditions: errc::not_enough_memory if there was a failure to allocate memory.
o2d_error::invalid_status if there was a failure to allocate a resource other than memory.

12.5.4 brush modifiers

void extend(experimental::io2d::extend e) noexcept;

 Effects: Sets the extend value for the brush object.
 Postconditions: _Extend == e.

void filter(experimental::io2d::filter f) noexcept;

 Effects: Sets the filter value for the brush object.
 Postconditions: _Filter == f.

void matrix(const matrix_2d& m) noexcept;

 Effects: Sets the matrix_2d value for the brush object.
 Postconditions: _Matrix == m.

12.5.5 brush observers

experimental::io2d::extend extend() const noexcept;

 Returns: _Extend.

experimental::io2d::filter filter() const noexcept;

 Returns: _Filter.

matrix_2d matrix() const noexcept;

 Returns: _Matrix.

brush_type type() const noexcept;

 Returns: _Brush_type.
12.6 Class solid_color_brush_factory

12.6.1 solid_color_brush_factory synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
   class solid_color_brush_factory {
      public:
         // 12.6.3, construct/copy/move/destroy:
         solid_color_brush_factory() noexcept;
         solid_color_brush_factory(const solid_color_brush_factory&) noexcept;
         solid_color_brush_factory& operator=(const solid_color_brush_factory&) noexcept;
         solid_color_brush_factory(solid_color_brush_factory&& other) noexcept;
         solid_color_brush_factory& operator=(solid_color_brush_factory&& other) noexcept;
         solid_color_brush_factory(const rgba_color& color) noexcept;

         // 12.6.4, modifiers:
         void color(const rgba_color& value) noexcept;

         // 12.6.5, observers:
         rgba_color color() const noexcept;

      private:
         rgba_color _Color; // exposition only
   } } } }

12.6.2 solid_color_brush_factory Description

The class solid_color_brush_factory describes a mutable factory for creating brush objects with uniform color and alpha data.

12.6.3 solid_color_brush_factory constructors and assignment operators

solid_color_brush_factory() noexcept;
   Effects: Constructs an object of type solid_color_brush_factory.
   Postcondition: _Color == rgba_color{}.

solid_color_brush_factory(const rgba_color& color) noexcept;
   Effects: Constructs an object of type solid_color_brush_factory. A brush created using this object will will have color as its color.
   Postcondition: _Color == color.

12.6.4 solid_color_brush_factory modifiers

void color(const rgba_color& value) noexcept;
   Effects: A brush created using this object will have value as its color.
   Postcondition: _Color == value.

12.6.5 solid_color_brush_factory observers

rgba_color color() const noexcept;
   Returns: _Color.
12.7 Class linear_brush_factory

12.7.1 linear_brush_factory synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  class linear_brush_factory {
public:
  // types
  typedef implementation-defined size_type; // See (12.7.2).

  // 12.7.4, construct/copy/move/destroy:
  linear_brush_factory() noexcept;
  linear_brush_factory(const linear_brush_factory&);
  linear_brush_factory& operator=(const linear_brush_factory&);
  linear_brush_factory(linear_brush_factory&& other) noexcept;
  linear_brush_factory& operator=(linear_brush_factory&& other) noexcept;
  linear_brush_factory(const vector_2d& begin, const vector_2d& end) noexcept;

  // 12.7.5, modifiers:
  void add_color_stop(double offset, const rgba_color& color);
  void add_color_stop(double offset, const rgba_color& color,
                       error_code& ec) noexcept;
  void color_stop(size_type index, double offset,
                  const rgba_color& color);
  void color_stop(size_type index, double offset,
                  const rgba_color& color, error_code& ec) noexcept;
  void begin_point(const vector_2d& value) noexcept;
  void end_point(const vector_2d& value) noexcept;

  // 12.7.6, observers:
  size_type color_stop_count() const noexcept;
  tuple<double, rgba_color> color_stop(size_type index) const;
  tuple<double, rgba_color> color_stop(size_type index,
                                       error_code& ec) const noexcept;
  vector_2d begin_point() const noexcept;
  vector_2d end_point() const noexcept;
private:
  vector_2d _Begin_point; // exposition only
  vector_2d _End_point; // exposition only
  vector<tuple<double, rgba_color>> _Color_stops; // exposition only
  }
}}}

12.7.2 linear_brush_factory::size_type

The type of linear_brush_factory::size_type shall comply with the restrictions specified for the size_type of the collection of color stops that is part of the observable state of a gradient (12.1.1).

12.7.3 linear_brush_factory Description

The class linear_brush_factory describes a mutable factory for creating brush objects with a linear gradient describing its color and alpha data.

2 For more information about gradients, including linear gradients, see 12.1.

12.7.4 linear_brush_factory constructors and assignment operators

§ 12.7.4
linear_brush_factory() noexcept;

Effects: Constructs an object of type linear_brush_factory.

Postconditions: _Begin_point == vector_2d{ }.
_End_point == vector_2d{ }.
_Color_stops.empty() == true.

linear_brush_factory(const vector_2d& begin, const vector_2d& end) noexcept;

Effects: Constructs an object of type linear_brush_factory.

Postconditions: _Begin_point == begin.
_End_point == end.
_Color_stops.empty() == true.

12.7.5 linear_brush_factory modifiers

void add_color_stop(double offset, const rgba_color& color);

Effects: Adds a color stop with an offset value of offset and a color value of color.

_Color_stops.push_back(make_tuple(offset, color)).

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if the attempt to add the color stop fails.

void add_color_stop(double offset, const rgba_color& color,
error_code& ec) noexcept;

void color_stop(size_type index, double offset,
const rgba_color& color);

Effects: Replaces the color stop at index index with a color stop with an offset of offset and a color of color.

Postconditions: _Color_stops.at(index) == make_tuple(offset, color).

Throws: As specified in Error reporting (3).

Error conditions: io2d_error::invalid_index if _Color_stops.size() <= index.

void begin_point(const vector_2d& value) noexcept;

Effects: Sets the begin point to value.

Postconditions: _Begin_point == value.

void end_point(const vector_2d& value) noexcept;

Effects: Sets the end point to value.

Postconditions: _End_point == value.
12.7.6 color_stop_count observers

size_type color_stop_count() const noexcept;

Returns: static_cast<size_type>(_Color_stops.size()).

tuple<double, rgba_color> color_stop(unsigned int index) const;

tuple<double, rgba_color> color_stop(unsigned int index,
    error_code& ec) const noexcept;

Returns: _Color_stops.at(index).

Throws: As specified in Error reporting (3).

Error conditions: io2d_error::invalid_index if _Color_stops.size() <= index.

vector_2d begin_point() const noexcept;

Returns: _Begin_point.

vector_2d end_point() const noexcept;

Returns: _End_point.

12.8 Class radial_brush_factory

12.8.1 radial_brush_factory synopsis

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

class radial_brush_factory {
    public:
    // types
    typedef implementation-defined size_type; // See (12.8.2).

    // 12.8.4, construct/copy/move/destroy:
    radial_brush_factory() noexcept;
    radial_brush_factory(const radial_brush_factory&);
    radial_brush_factory& operator=(const radial_brush_factory&);
    radial_brush_factory(radial_brush_factory&& other) noexcept;
    radial_brush_factory& operator=(radial_brush_factory&& other) noexcept;
    radial_brush_factory(const vector_2d& center0, double radius0,
        const vector_2d& center1, double radius1) noexcept;

    // 12.8.5, modifiers:
    void add_color_stop(double offset, const rgba_color& color);
    void add_color_stop(double offset, const rgba_color& color,
        error_code& ec) noexcept;
    void color_stop(size_type index, double offset, const rgba_color& color);
    void color_stop(size_type index, double offset, const rgba_color& color,
        error_code& ec) noexcept;
    void radial_circles(const vector_2d& center0, double radius0,
        const vector_2d& center1, double radius1) noexcept;

    // 12.8.6, observers:
    size_type color_stop_count() const noexcept;
    tuple<double, rgba_color> color_stop(size_type index) const;
    tuple<double, rgba_color> color_stop(size_type index,
        error_code& ec) const noexcept;
    tuple<vector_2d, double, vector_2d, double> radial_circles() const noexcept;

§ 12.8.1
private:
    vector_2d _Center0; // exposition only
    double _Radius0;    // exposition only
    vector_2d _Center1; // exposition only
    double _Radius1;    // exposition only
    vector<tuple<double, rgba_color>> _Color_stops; // exposition only
};

12.8.2 radial_brush_factory::size_type

The type of radial_brush_factory::size_type shall comply with the restrictions specified for the size_type of the collection of color stops that is part of the observable state of a gradient (12.1.1).

12.8.3 radial_brush_factory Description

The class radial_brush_factory describes a mutable factory for creating brush objects with a radial gradient describing its color and alpha data.

1 For more information about gradients, including radial gradients, see 12.1.

12.8.4 radial_brush_factory constructors and assignment operators

radial_brush_factory() noexcept;

Effects: Constructs an object of type radial_brush_factory.

Postconditions: _Center0 == vector_2d{}. _Radius0 == 0.0.
_Center1 == vector_2d{}. _Radius1 == 0.0.
_Color_stops.empty() == true.

radial_brush_factory(const vector_2d& center0, double radius0,
const vector_2d& center1, double radius1) noexcept;

Effects: Constructs an object of type radial_brush_factory.

Postconditions: _Center0 == center0. 
_Radius0 == radius0. 
_Center1 == center1. 
_Radius1 == radius1. 
_Color_stops.empty() == true.

12.8.5 radial_brush_factory modifiers

void add_color_stop(double offset, const rgba_color& color);
void add_color_stop(double offset, const rgba_color& color,
error_code& ec) noexcept;

Effects: Adds a color stop with an offset value of offset and a color value of color.

Postconditions: _Color_stops.push_back(make_tuple(offset, color)).

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if the attempt to add the color stop fails.
void color_stop(size_type index, double offset, const rgba_color& color);
void color_stop(size_type index, double offset, const rgba_color& color,
    error_code& ec) noexcept;

Effects: Replaces the color stop at index index with a color stop with an offset of offset and a color of color.
Postconditions: _Color_stops.at(index) == make_tuple(offset, color).

 Throws: As specified in Error reporting (3).

Error conditions: io2d_error::invalid_index if _Color_stops.size() <= index.

void radial_circles(const vector_2d& center0, double radius0,
    const vector_2d& center1, double radius1) noexcept;

Postconditions: _Center0 == center0.
_Radius0 == radius0.
_Center1 == center1.
_Radius1 == radius1.

12.8.6 radial_brush_factory observers

size_type color_stop_count() const noexcept;

Returns: static_cast<size_type>(_Color_stops.size()).

tuple<double, rgba_color> color_stop(size_type index) const;
tuple<double, rgba_color> color_stop(size_type index,
    error_code& ec) const noexcept;

Returns: _Color_stops.at(index).

Throws: As specified in Error reporting (3).

Error conditions: io2d_error::invalid_index if _Color_stops.size() <= index.

tuple<vector_2d, double, vector_2d, double> radial_circles() const noexcept;

Returns: make_tuple(_Center0, _Radius0, _Center1, _Radius1).

12.9 Class surface_brush_factory

12.9.1 surface_brush_factory synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
class surface_brush_factory {
public:

// 12.9.3, construct/copy/move/destroy:
surface_brush_factory() noexcept;
surface_brush_factory(const surface_brush_factory&) = delete;
surface_brush_factory& operator=(const surface_brush_factory&) = delete;
surface_brush_factory(surface_brush_factory& other) noexcept;
surface_brush_factory& operator=(surface_brush_factory& other) noexcept;
surface_brush_factory(experimental::io2d::surface& s);
surface_brush_factory(experimental::io2d::surface& s, error_code& ec) noexcept;

// 12.9.4, modifiers:
image_surface surface(experimental::io2d::surface& s);
void surface(experimental::io2d::surface& s, image_surface oldSurface, error_code& ec) noexcept;

§ 12.9.1 117
void surface(experimental::io2d::surface& s, error_code& ec) noexcept;

// 12.9.5, observers:
bool has_surface() const noexcept;
const image_surface& surface() const;

private:
    unique_ptr<image_surface> _Surface; // exposition only
};

12.9.2 surface_brush_factory Description  [surfacebrushfact.intro]
The class surface_brush_factory describes a mutable factory for creating brush objects with a surface object describing its color and alpha data.

12.9.3 surface_brush_factory constructors and assignment operators  [surfacebrushfact.cons]
surface_brush_factory(experimental::io2d::surface& s);
surface_brush_factory(experimental::io2d::surface& s, error_code& ec) noexcept;

1 Effects: Constructs an object of type surface_brush_factory. Calls s.flush() then creates a copy of s and stores it as the surface which will be painted by a brush formed using *this.

2 Postconditions: _Surface contains a valid pointer to an image_surface object with the same width, height, format as s and a copy of the visual data of s as raster graphics data.

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

4 Remarks: Excluding any effects of calling s.flush(), s shall not be modified.

5 Error conditions: Any error condition documented for surface::flush.

6 errc::invalid_argument if !s.has_surface_resource().

7 io2d_error::surface_finished if s.is_finished().

8 errc::not_enough_memory if an attempted memory allocation failed.

12.9.4 surface_brush_factory modifiers  [surfacebrushfact.modifiers]
void surface(experimental::io2d::surface& s);
void surface(experimental::io2d::surface& s, error_code& ec) noexcept;

1 Effects: Calls s.flush() then creates a copy of s and stores it as the surface which will be painted by a brush formed using *this. The stored surface shall be accessible as an object of type image_surface.

2 Postconditions: _Surface contains a valid pointer to an image_surface object with the same width, height, format as s and a copy of the visual data of s as raster graphics data.

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

4 Remarks: Excluding the effects of calling s.flush(), s shall not be modified.

5 Error conditions: Any error condition documented for surface::flush (13.10).

6 errc::invalid_argument if !s.has_surface_resource().

7 io2d_error::surface_finished if s.is_finished().

8 errc::not_enough_memory if an attempted memory allocation failed.
12.9.5 surface_brush_factory observers

bool has_surface() const noexcept;

1 \textit{Returns}: \_Surface.get \neq \texttt{nullptr}.

const image_surface& surface() const;

2 \textit{Requires}: has_surface() == true.

3 \textit{Returns}: \_Surface.get().
13 Surfaces [surfaces]

Surfaces are the platform upon which all rendering and composing operations take place. The various surface classes provide specific functionality that enables a broad variety of tasks to be accomplished in efficient ways.

This section is forthcoming in a future revision.

13.1 Enum class antialias [antialias]

13.1.1 antialias Summary [antialias.summary]

The antialias enum class specifies the type of anti-aliasing that the rendering system shall use for rendering text. See Table 10 for the meaning of each antialias enumerator.

13.1.2 antialias Synopsis [antialias.synopsis]

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

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

13.1.3 antialias Enumerators [antialias.enumerators]

Table 10 — antialias enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_antialias</td>
<td>The meaning of this value is implementation-defined.</td>
</tr>
<tr>
<td>none</td>
<td>No anti-aliasing.</td>
</tr>
<tr>
<td>gray</td>
<td>Monochromatic anti-aliasing. [Note: When rendering black text on a white background, this would produce gray-scale.</td>
</tr>
<tr>
<td>subpixel</td>
<td>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.</td>
</tr>
<tr>
<td>fast</td>
<td>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]</td>
</tr>
</tbody>
</table>
Table 10 — antialias enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>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.)</td>
</tr>
<tr>
<td>best</td>
<td>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.)</td>
</tr>
</tbody>
</table>

13.2 Enum class content

13.2.1 content Summary

The content enum class describes the type of data that a surface object contains. See Table 11 for the meaning of each enumerator.

13.2.2 content Synopsis

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

13.2.3 content Enumerators

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>The surface holds opaque color data only.</td>
</tr>
<tr>
<td>alpha</td>
<td>The surface holds alpha (translucency) data only.</td>
</tr>
<tr>
<td>color_alpha</td>
<td>The surface holds both color and alpha data.</td>
</tr>
</tbody>
</table>

13.3 Enum class fill_rule

13.3.1 fill_rule Summary

The fill_rule enum class determines how the individuals paths in path objects are filled. For information about multiple paths in a single path object, see the description of the path class (10.4).

In order to determine whether a point will be included in a fill operation, create a ray from the point to infinity. The direction of the ray does not matter provided that it does not pass through the end point of a path segment and does not intersect the path at a point that is tangent to the path. The intersections of the ray with the path are used in conjunction with the current fill_rule value to determine whether a point is filled. See Table 12 for the meaning of each fill_rule enumerator. (Note: When used below, if the term path is formatted normally, i.e. path, it refers to an individual path within a path object. If the

§ 13.3.1
term is formatted in code style, i.e. `path`, it refers to the `path` type, which may contain one or more paths.

— end note —

### 13.3.2 fill_rule Synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class fill_rule {
    winding,
    even_odd
  };
} } } }

### 13.3.3 fill_rule Enumerators

Table 12 — fill_rule enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>winding</td>
<td>Starting with a count of zero, whenever the path crosses the ray from left to right, add one to the count, and whenever the path crosses the ray from right to left, subtract one from the count. If the count is zero the point is not filled, otherwise it is filled.</td>
</tr>
<tr>
<td>even_odd</td>
<td>Count the number of times the ray intersects the path. If the count is even, the point is not filled, otherwise it is filled.</td>
</tr>
</tbody>
</table>

### 13.4 Enum class line_cap

#### 13.4.1 line_cap Summary

The `line_cap` enum class specifies how the ends of lines should be rendered when a `path` object is stroked. See Table 13 for the meaning of each `line_cap` enumerator.

### 13.4.2 line_cap Synopsis

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

### 13.4.3 line_cap Enumerators

Table 13 — line_cap enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>butt</td>
<td>The line has no cap. It terminates exactly at the end point.</td>
</tr>
<tr>
<td>round</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 13 — line_cap enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>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.</td>
</tr>
</tbody>
</table>

13.5 Enum class line_join

13.5.1 line_join Summary

The line_join enum class specifies how the junction of two line segments should be rendered when a path is stroked. See Table 14 for the meaning of each enumerator.

13.5.2 line_join Synopsis

```cpp
namespace std { namespace experimental { namespace drawing { inline namespace v1 {
    enum class line_join {
        miter,
        round,
        bevel
    };
} } } }
```

13.5.3 line_join Enumerators

Table 14 — line_join enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>miter</td>
<td>Joins will be mitered or beveled, depending on the current Miter Limit (13.10.3).</td>
</tr>
<tr>
<td>round</td>
<td>Joins will be rounded, with the center of the circle being the join point.</td>
</tr>
<tr>
<td>bevel</td>
<td>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.</td>
</tr>
</tbody>
</table>

13.6 Enum class compositing_operator

13.6.1 compositing_operator Summary

The compositing_operator enum class specifies composition algorithms. See Table 15, Table 16 and Table 17 for the meaning of each compositing_operator enumerator.

13.6.2 compositing_operator Synopsis

```cpp
namespace std { namespace experimental { namespace drawing { inline namespace v1 {
    enum class compositing_operator {
        // basic
        over,
        clear,
    };
} } } }
```

§ 13.6.2
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
};
} } } }

13.6.3 compositing_operator Enumerators [compositing.operator.enumerators]

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].

Where a pixel format for a pixel has no alpha channel, the pixel format shall be treated as though it had an alpha channel with a value of 1.0 for purposes of evaluating the formulas.

Where a pixel format for a pixel has no color channels, the pixel format shall be treated as though it had a value of 0.0 for all color channels for purposes of evaluating the formulas.

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

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

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

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

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

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]

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

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

The basic enumerators specify a value for Bound. This value may be 'Yes', 'No', or 'N/A'.

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.

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.

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.

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.

If a clip is set and the Bound value is 'No', then only those areas of the surface that 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 15 — \texttt{compositing\textunderscore operator} basic enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Bound</th>
<th>Color</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>Yes</td>
<td>( Rc = 0 )</td>
<td>( Ra = 0 )</td>
</tr>
<tr>
<td>source</td>
<td>Yes</td>
<td>( Rc = Sc )</td>
<td>( Ra = Sa )</td>
</tr>
<tr>
<td>over</td>
<td>N/A</td>
<td>( Rc = \frac{(Sca + Dca \times (1 - Sa))}{Ra} )</td>
<td>( Ra = Sa + Da \times (1 - Sa) )</td>
</tr>
<tr>
<td>in</td>
<td>No</td>
<td>( Rc = Sc )</td>
<td>( Ra = Sa \times Da )</td>
</tr>
<tr>
<td>out</td>
<td>No</td>
<td>( Rc = Sc )</td>
<td>( Ra = Sa \times (1 - Da) )</td>
</tr>
<tr>
<td>atop</td>
<td>N/A</td>
<td>( Rc = Sca + Dc \times (1 - Sa) )</td>
<td>( Ra = Da )</td>
</tr>
</tbody>
</table>

\section*{§ 13.6.3}
Table 15 — `compositing_operator` basic enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Bound</th>
<th>Color</th>
<th>Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>N/A</td>
<td>$R_c = D_c$</td>
<td>$R_a = D_a$</td>
</tr>
<tr>
<td>dest_over</td>
<td>N/A</td>
<td>$R_c = \frac{(S_c \times (1 - D_a) + D_c a)}{R_a}$</td>
<td>$R_a = (1 - D_a) \times S_a + D_a$</td>
</tr>
<tr>
<td>dest_in</td>
<td>No</td>
<td>$R_c = D_c$</td>
<td>$R_a = S_a \times D_a$</td>
</tr>
<tr>
<td>dest_out</td>
<td>N/A</td>
<td>$R_c = D_c$</td>
<td>$R_a = (1 - S_a) \times D_a$</td>
</tr>
<tr>
<td>dest_atop</td>
<td>No</td>
<td>$R_c = S_c \times (1 - D_a) + D_c a$</td>
<td>$R_a = S_a$</td>
</tr>
<tr>
<td>xor_op</td>
<td>N/A</td>
<td>$R_c = \frac{(S_c \times (1 - D_a) + D_c a \times (1 - S_a))}{R_a}$</td>
<td>$R_a = S_a + D_a - 2 \times S_a \times D_a$</td>
</tr>
<tr>
<td>add</td>
<td>N/A</td>
<td>$R_c = \frac{(S_c + D_c a)}{R_a}$</td>
<td>$R_a = \min(1, S_a + D_a)$</td>
</tr>
<tr>
<td>saturate</td>
<td>N/A</td>
<td>$R_c = \frac{(\min(S_a, 1 - D_a) \times S_c + D_c a)}{R_a}$</td>
<td>$R_a = \min(1, S_a + D_a)$</td>
</tr>
</tbody>
</table>

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: $R_c = \frac{1}{R_a} \times ((1 - D_a) \times S_c + (1 - S_a) \times D_c a + S_a \times D_a \times f(S_c, D_c))$. The function $f(S_c, D_c)$ is the component of the formula that is enumerator dependent.

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.

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]

For the enumerators in Table 16 and Table 17 the result color’s alpha channel value formula is as follows: $R_a = S_a + D_a \times (1 - S_a)$. [Note: Since it is the same formula for all enumerators in those tables, the formula is not included in those tables. — end note]

All of the blend enumerators and hsl enumerators have a Bound value of ‘N/A’.

Table 16 — `compositing_operator` blend enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiply</td>
<td>$f(S_c, D_c) = S_c \times D_c$</td>
</tr>
<tr>
<td>screen</td>
<td>$f(S_c, D_c) = S_c + D_c - S_c \times D_c$</td>
</tr>
</tbody>
</table>

§ 13.6.3
<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>overlay</strong></td>
<td>$f(Sc, Dc) = 2 \times Sc \times Dc$ if $Dc \leq 0.5$</td>
</tr>
<tr>
<td></td>
<td>else $f(Sc, Dc) = 1 - 2 \times (1 - Sc) \times (1 - Dc)$</td>
</tr>
<tr>
<td><strong>Note:</strong> The difference between this enumerator and <strong>hard_light</strong> is that this tests the destination color ($Dc$) whereas <strong>hard_light</strong> tests the source color ($Sc$). — end note</td>
<td></td>
</tr>
<tr>
<td><strong>darken</strong></td>
<td>$f(Sc, Dc) = \min(Sc, Dc)$</td>
</tr>
<tr>
<td><strong>lighten</strong></td>
<td>$f(Sc, Dc) = \max(Sc, Dc)$</td>
</tr>
<tr>
<td><strong>color_dodge</strong></td>
<td>$f(Sc, Dc) = \min(1, \frac{Dc}{1 - Sc})$ if $Dc &lt; 1$</td>
</tr>
<tr>
<td></td>
<td>else $f(Sc, Dc) = 1$</td>
</tr>
<tr>
<td><strong>color_burn</strong></td>
<td>$f(Sc, Dc) = 1 - \min(1, \frac{1 - Dc}{Sc})$ if $Dc &gt; 0$</td>
</tr>
<tr>
<td></td>
<td>else $f(Sc, Dc) = 0$</td>
</tr>
<tr>
<td><strong>hard_light</strong></td>
<td>$f(Sc, Dc) = 2 \times Sc \times Dc$ if $Sc \leq 0.5$</td>
</tr>
<tr>
<td></td>
<td>else $f(Sc, Dc) = 1 - 2 \times (1 - Sc) \times (1 - Dc)$</td>
</tr>
</tbody>
</table>
Table 16 — \texttt{compositing\_operator} blend enumerator meanings
(continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Color</th>
</tr>
</thead>
</table>
| soft\_light   | $\begin{cases} f(Sc, Dc) = Dc - (1 - 2 \times Sc) \times Dc \times (1 - Dc) \\
|               | \text{if} (Sc \leq 0.5) \\
|               | \text{else} \{ f(Sc, Dc) = Dc + (2 \times Sc - 1) \times (g(Dc) - Sc) \} \end{cases}$ |
|               | $g(Dc)$ is defined as follows: $\begin{cases} g(Dc) = \\
|               | \text{if} (Dc \leq 0.25) \{ g(Dc) = ((16 \times Dc - 12) \times Dc + 4) \times Dc \} \text{else} \{ g(Dc) = \sqrt{Dc} \} \end{cases}$ |
| difference    | $f(Sc, Dc) = \text{abs}(Dc - Sc)$                                      |
| exclusion     | $f(Sc, Dc) = Sc + Dc - 2 \times Sc \times Dc$                        |

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).

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

- $\text{min}(x, y, z) = \text{min}(x, \text{min}(y, z))$
- $\text{max}(x, y, z) = \text{max}(x, \text{max}(y, z))$
- $\text{sat}(C) = \text{max}(Cr, Cg, Cb) - \text{min}(Cr, Cg, Cb)$
- $\text{lum}(C) = Cr \times 0.3 + Cg \times 0.59 +Cb \times 0.11$

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

§ 13.6.3
\[
\begin{align*}
Cr &= L + \frac{((Cr - L) \times (1 - L))}{(X - L)} \\
Cg &= L + \frac{((Cg - L) \times (1 - L))}{(X - L)} \\
Cb &= L + \frac{((Cb - L) \times (1 - L))}{(X - L)} \\
\end{align*}
\]

return \(C\)

\[
\begin{align*}
 LD &= L - \text{lum}(C) \\
Cr &= Cr + D \\
Cg &= Cg + D \\
Cb &= Cb + D \\
\text{return clip\_color}(C) \\
\end{align*}
\]

\[
\begin{align*}
R &= C \\
\text{auto\& max} &= (Rr > Rg) ? ((Rr > Rb) ? Rr : Rb) : ((Rg > Rb) ? Rg : Rb) \\
\text{auto\& mid} &= (Rr > Rg) ? ((Rr > Rb) ? (Rg > Rb) ? Rg : Rb) : Rr \\
\text{auto\& min} &= (Rr > Rg) ? ((Rr > Rb) ? (Rg > Rb) ? Rg : Rb) : Rr \\
\text{if } (\text{max} \geq \text{min}) \{ \\
  \text{mid} &= \frac{(\text{mid} - \text{min}) \times S}{\text{max} - \text{min}} \\
  \text{max} &= S \\
\} \\
\text{else} \{ \\
  \text{mid} &= 0.0 \\
  \text{max} &= 0.0 \\
\} \\
\text{min} &= 0.0 \\
\text{return } R \\
\end{align*}
\]

[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 17 — compositing\_operator hsl enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Color &amp; Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsl_hue</td>
<td>(f(Sc, Dc) = \text{set_lum}(\text{set_sat}(Sc, \text{sat}(Dc)), \text{lum}(Dc)))</td>
</tr>
<tr>
<td>hsl_saturation</td>
<td>(\text{set_lum}(\text{set_sat}(Dc, \text{sat}(Sc)), \text{lum}(Dc)))</td>
</tr>
<tr>
<td>hsl_color</td>
<td>(f(Sc, Dc) = \text{set_lum}(Sc, \text{lum}(Dc)))</td>
</tr>
<tr>
<td>hsl_luminosity</td>
<td>(f(Sc, Dc) = \text{set_lum}(Dc, \text{lum}(Sc)))</td>
</tr>
</tbody>
</table>

### 13.7 Enum class format

#### 13.7.1 format Summary

The \text{format} enum class indicates a visual data format. See Table 18 for the meaning of each \text{format} enumerator.
Unless otherwise specified, a visual data format shall be an unsigned integral value of the specified bit size in native-endian format.

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 \( \text{red} \), 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]

### 13.7.2 format Synopsis

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

### 13.7.3 format Enumerators

Table 18 — format enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid</td>
<td>A previously specified format is unsupported by the implementation.</td>
</tr>
<tr>
<td>argb32</td>
<td>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.</td>
</tr>
<tr>
<td>rgb24</td>
<td>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.</td>
</tr>
<tr>
<td>a8</td>
<td>An 8-bit transparency data pixel format. All 8 bits are an alpha channel.</td>
</tr>
<tr>
<td>rgb16_565</td>
<td>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.</td>
</tr>
<tr>
<td>rgb30</td>
<td>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.</td>
</tr>
</tbody>
</table>
13.8 Enum class scaling

13.8.1 scaling Summary

1 The scaling enum class specifies the type of scaling a display_surface will use when the size of its Display Buffer (13.12.2) differs from the size of its Back Buffer (13.12.2).

2 See Table 19 for the meaning of each scaling enumerator.

13.8.2 scaling Synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 {
  enum class scaling {
    letterbox, uniform, fill_uniform, fill_exact, none
  }
} } } }

13.8.3 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 (13.12.2) 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 1.3.

The Display Buffer (13.12.2) of ds is 1280x720, giving it an aspect ratio of 1.7.

When a rectangle is defined in an example, the coordinate \((x_1, y_1)\) denotes the top left corner of the rectangle, inclusive, and the coordinate \((x_2, y_2)\) denotes the bottom right corner of the rectangle, exclusive. As such, a rectangle with \((x_1, y_1) = (10, 10)\) and \((x_2, y_2) = (20, 20)\) 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 19 — scaling enumerator meanings

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>letterbox</td>
<td>Fill the Display Buffer with the Letterbox Brush (13.12.3) 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]</td>
</tr>
</tbody>
</table>
Table 19 — scaling enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 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 transferred to the Display Buffer using sampling such that it is in the rectangle 
  \[(x_1, y_1) = \left(\frac{1280}{2} - \frac{960}{2}, 0\right) = (160, 0),\] 
  \[(x_2, y_2) = (960 + \left(\frac{1280}{2} - \frac{960}{2}\right), 720) = (1120, 720).\] 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 19 — scaling enumerator meanings (continued)

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 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 \((x_1, y_1) = (0, -120), (x_2, y_2) = (1280, 840)\). 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). ]
| 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 \((x_1, y_1) = (0, 0), (x_2, y_2) = (1280, 720)\). 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. ]
| 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 \((x_1, y_1) = (0, 0), (x_2, y_2) = (640, 480)\) such that no scaling occurs and the origin of the Back Buffer is at the origin of the Display Buffer. ]

13.9 Class device

13.9.1 device synopsis

namespace std { namespace experimental { namespace io2d { inline namespace v1 { public device {
// 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);

// 13.9.3, modifiers:
void flush() noexcept;
void lock();
void lock(error_code& ec) noexcept;
void unlock();
void unlock(error_code& ec) noexcept;
}
} } } }

13.9.2 device Description

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

2 A device object is obtained from a surface or surface-derived object.

13.9.3 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.

```cpp
void lock();
void lock(error_code& ec) noexcept;
```

**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.

**Throws:** As described in Error reporting (3).

**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]

```cpp
void unlock();
void unlock(error_code& ec) noexcept;
```

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

**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.

**Throws:** As described in Error reporting (3).

**Remarks:** This function shall not be called more times than `lock` has been called; no diagnostic is
required.

### 13.10 Class surface

A rendering and composing operation is an operation that is either a single-phase operation consisting of a
composing operation or a multi-phase operation consisting of a rendering operation followed by a composing
operation.

### 13.10.1 surface synopsis

```cpp
definitions
namespace std { namespace experimental { namespace io2d { inline namespace v1 {
class surface {
public:
    // 13.10.8, 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();

    // 13.10.9, state modifiers:
    virtual 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);
virtual void save();
virtual void save(error_code& ec) noexcept;
virtual void restore();
virtual void restore(error_code& ec) noexcept;
void brush(experimental::nullopt_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::nullopt_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 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::nullopt_t) noexcept;
void path(const experimental::io2d::path& p);
void path(const experimental::io2d::path& p, error_code& ec) noexcept;

// 13.10.10, immediate path modifiers:
experimental::io2d::path_factory& immediate() noexcept;

// 13.10.11, 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 vector_2d& o = vector_2d{ }, 
    extend e = extend::none, filter f = filter::good);
void fill(const surface& s, error_code& ec, 
    const vector_2d& o = vector_2d{ }, extend e = extend::none, 
    filter f = filter::good) noexcept;
void fill(const surface& s, const matrix_2d& m, extend e = extend::none, 
    filter f = filter::good);
void fill(const surface& s, const matrix_2d& m, error_code& ec, 
    extend e = extend::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 vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
void fill_immediate(const surface& s, error_code& ec,
    const vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good) noexcept;
void fill_immediate(const surface& s, const matrix_2d& m,
    extend e = extend::none, filter f = filter::good);
void fill_immediate(const surface& s, const matrix_2d& m, error_code& ec,
    extend e = extend::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 vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
void paint(const surface& s, error_code& ec,
    const vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good) noexcept;
void paint(const surface& s, const matrix_2d& m,
    extend e = extend::none, filter f = filter::good);
void paint(const surface& s, error_code& ec,
    const matrix_2d& m, extend e = extend::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 vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good);
void paint(const surface& s, double alpha, error_code& ec,
    const vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good) noexcept;
void paint(const surface& s, double alpha, const matrix_2d& m,
    extend e = extend::none, filter f = filter::good);
void paint(const surface& s, double alpha, const matrix_2d& m,
    error_code& ec, extend e = extend::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 vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
void stroke(const surface& s, error_code& ec,
const vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good) noexcept;
void stroke(const surface& s, const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void stroke(const surface& s, const matrix_2d& m, error_code& ec, extend e = extend::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 vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good);
void stroke_immediate(const surface& s, error_code& ec, const vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good) noexcept;
void stroke_immediate(const surface& s, const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void stroke_immediate(const surface& s, const matrix_2d& m, error_code& ec, extend e = extend::none, filter f = filter::good) noexcept;
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 vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good);
void mask(const experimental::io2d::brush& mb, const surface& s, error_code& ec, const vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good) noexcept;
void mask(const experimental::io2d::brush& mb, const surface& s, const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void mask(const experimental::io2d::brush& mb, const surface& s, const matrix_2d& m, error_code& ec, extend e = extend::none, filter f = filter::good) noexcept;
void mask(const surface& ms);
void mask(const surface& ms, error_code& ec) noexcept;
void mask(const surface& ms, const rgba_color& c);
void mask(const surface& ms, const rgba_color& c, error_code& ec) noexcept;
void mask(const surface& ms, const experimental::io2d::brush& b);
void mask(const surface& ms, const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask(const surface& ms, const surface& s, const vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good);
void mask(const surface& ms, const surface& s, error_code& ec, const vector_2d& o = vector_2d{}, extend e = extend::none, filter f = filter::good) noexcept;
void mask(const surface& ms, const surface& s, const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void mask(const surface& ms, const surface& s, const matrix_2d& m, error_code& ec, extend e = extend::none, filter f = filter::good) noexcept;
void mask(const surface& ms, const vector_2d& mo);
void mask(const surface& ms, const vector_2d& mo, error_code& ec) noexcept;
void mask(const surface& ms, const vector_2d& mo, const rgba_color& c);
void mask(const surface& ms, const vector_2d& mo, const rgba_color& c, error_code& ec) noexcept;
void mask(const surface& ms, const vector_2d& mo, const experimental::io2d::brush& b);
void mask(const surface& ms, const vector_2d& mo, const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask(const surface& ms, const vector_2d& mo, const surface& s, const vector_2d& o = vector_2d{}, extend e = extend::none, filter f = filter::good);
void mask(const surface& ms, const vector_2d& mo, const surface& s, error_code& ec, const vector_2d& o = vector_2d{}, extend e = extend::none, 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 vector_2d& o = vector_2d{}, extend e = extend::none, filter f = filter::good);
void mask_immediate(const experimental::io2d::brush& mb, const surface& s, error_code& ec, const vector_2d& o = vector_2d{}, extend e = extend::none, filter f = filter::good) noexcept;
void mask_immediate(const experimental::io2d::brush& mb, const surface& s, const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void mask_immediate(const experimental::io2d::brush& mb, const surface& s, const matrix_2d& m, error_code& ec, extend e = extend::none, filter f = filter::good) noexcept;
void mask_immediate(const surface& ms);
void mask_immediate(const surface& ms, error_code& ec) noexcept;
void mask_immediate(const surface& ms, const rgba_color& c);
void mask_immediate(const surface& ms, const rgba_color& c, error_code& ec) noexcept;

§ 13.10.1
void mask_immediate(const surface& ms, const experimental::io2d::brush& b);
void mask_immediate(const surface& ms, const experimental::io2d::brush& b,
    error_code& ec) noexcept;
void mask_immediate(const surface& ms, const surface& s,
    const vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good);
void mask_immediate(const surface& ms, const surface& s, error_code& ec,
    const vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good) noexcept;
void mask_immediate(const surface& ms, const surface& s, const matrix_2d& m,
    extend e = extend::none, filter f = filter::good);
void mask_immediate(const surface& ms, const surface& s,
    const matrix_2d& m, error_code& ec, extend e = extend::none,
    filter f = filter::good) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const rgba_color& c);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const rgba_color& c, error_code& ec) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const experimental::io2d::brush& b);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const surface& s, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const surface& s, error_code& ec, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const surface& s, const matrix_2d& m, extend e = extend::none,
    filter f = filter::good);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const surface& s, const matrix_2d& m, error_code& ec,
    extend e = extend::none, filter f = filter::good) noexcept;

// 13.10.13, text render modifiers:
vector_2d render_text(const string& utf8, const vector_2d& pos);
vector_2d render_text(const string& utf8, const vector_2d& pos,
    error_code& ec) noexcept;
vector_2d render_text(const string& utf8, const vector_2d& pos,
    const rgba_color& c);
vector_2d render_text(const string& utf8, const vector_2d& pos,
    const rgba_color& c, error_code& ec) noexcept;
vector_2d render_text(const string& utf8, const vector_2d& pos,
    const experimental::io2d::brush& b);
vector_2d render_text(const string& utf8, const vector_2d& pos,
    const experimental::io2d::brush& b, error_code& ec) noexcept;
vector_2d render_text(const string& utf8, const vector_2d& pos,
    const surface& s, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
vector_2d render_text(const string& utf8, const vector_2d& pos,
    const surface& s, error_code& ec, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
extend e = extend::none, filter f = filter::good) noexcept;
vector_2d render_text(const string& utf8, const vector_2d& pos,
const surface& s, const matrix_2d& m, extend e = extend::none,
filter f = filter::good);
vector_2d render_text(const string& utf8, const vector_2d& pos,
const surface& s, const matrix_2d& m, error_code& ec,
extend e = extend::none, filter f = filter::good) noexcept;

// 13.10.14, transformation modifiers:
void matrix(const matrix_2d& matrix);
void matrix(const matrix_2d& matrix, error_code& ec) noexcept;

// 13.10.15, font modifiers:
void font_face(const string& typeface, font_slant sl, font_weight w);
void font_face(const string& typeface, font_slant sl, font_weight w,
error_code& ec) noexcept;
void font_face(const experimental::io2d::font_face& f);
void font_face(const experimental::io2d::font_face& f, error_code& ec)
noexcept;
void font_size(double s) noexcept;
void font_matrix(const matrix_2d& m);
void font_matrix(const matrix_2d& m, error_code& ec) noexcept;
void font_options(const font_options& fo) noexcept;

// 13.10.16, 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;

// 13.10.17, 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(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;
The `surface` class provides an interface for managing a graphics data graphics resource and its observable state data (13.10.3).

A `surface` object is a move-only object.

The `surface` class provides two ways to modify its graphics resource:

- Rendering and composing operations.
- Mapping.

[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]

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.

[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]

The definitions of the rendering and composing operations in 13.10.4 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.

The definitions of the rendering and composing operations in 13.10.4 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.

13.10.3 surface state

Table 20 specifies the name, type, function, and default value for each item of a surface’s observable state.
13.10.3.1 surface state default values

Table 20 — Surface observable state

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Brush</td>
<td>brush</td>
<td>This is the brush that shall be available for use when performing rendering and composing operations</td>
<td>brush{ rgba_color::black() }</td>
</tr>
<tr>
<td>General Antialiasing</td>
<td>antialias</td>
<td>This is the type of antialiasing that should be used when performing non-text rendering operations</td>
<td>antialias::default_-antialias</td>
</tr>
<tr>
<td>Dash Pattern</td>
<td>dashes</td>
<td>This specifies the pattern that shall be used when performing stroke rendering operations</td>
<td>dashes{ vector&lt;double&gt;(), 0.0 }</td>
</tr>
<tr>
<td>Fill Rule</td>
<td>fill_rule</td>
<td>This controls which areas of paths shall be eligible to be filled when performing fill rendering operations</td>
<td>fill_rule::winding</td>
</tr>
<tr>
<td>Line Cap</td>
<td>line_cap</td>
<td>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</td>
<td>line_cap::butt</td>
</tr>
<tr>
<td>Line Join</td>
<td>line_join</td>
<td>This specifies how the join point of two path segments that are joined to each other shall be rendered when performing stroke rendering operations</td>
<td>line_join::miter</td>
</tr>
<tr>
<td>Line Width</td>
<td>double</td>
<td>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</td>
<td>2.0</td>
</tr>
<tr>
<td>Miter Limit</td>
<td>double</td>
<td>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.</td>
<td>10.0</td>
</tr>
<tr>
<td>Composition Operator</td>
<td>compositing_-operator</td>
<td>This specifies the composition algorithm that shall be used when performing rendering operations</td>
<td>compositing_-operator::over</td>
</tr>
<tr>
<td>Clip Area</td>
<td>unspecified</td>
<td>The areas of the surface which shall be the only areas in which composing operations may have any effect</td>
<td>An area which contains the entire area of the surface</td>
</tr>
</tbody>
</table>

§ 13.10.3.1
Table 20 — Surface observable state (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Current Path</strong></td>
<td>path</td>
<td>This is the path object that shall be used when performing non-immediate rendering operations</td>
<td>path{ path_factory{ } }</td>
</tr>
<tr>
<td><strong>Immediate Path</strong></td>
<td>path_factory</td>
<td>This is the path_factory object that shall be used when performing immediate rendering operations</td>
<td>path_factory{ }</td>
</tr>
<tr>
<td><strong>Transformation Matrix</strong></td>
<td>matrix_2d</td>
<td>This is the matrix_2d object that shall be used to transform coordinates from an object’s local coordinate space to the surface’s coordinate space when performing rendering operations</td>
<td>matrix_2d::init_identity{ }</td>
</tr>
<tr>
<td><strong>Font Face</strong></td>
<td>font_face</td>
<td>This is the font that shall be used when performing text rendering operations</td>
<td>implementation-defined</td>
</tr>
<tr>
<td><strong>Font Matrix</strong></td>
<td>matrix_2d</td>
<td>This is the matrix_2d object that shall be used to transform coordinates from a font’s local coordinate space to the surface’s coordinate space when performing text rendering operations</td>
<td>matrix_2d::init_scale{ { 10.0, 10.0 } }</td>
</tr>
<tr>
<td><strong>Font Options</strong></td>
<td>font_options</td>
<td>This is the font_options object that shall be used to determine certain aspects of how a font should be rendered when performing text rendering operations</td>
<td>font_option{ antialias::default_antialias, subpixel_order::default_subpixel_order }</td>
</tr>
</tbody>
</table>

1 [Note: The Clip Area may be modified by intersecting the current Clip Area with the areas of a path 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 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]

13.10.3.2 surface saved state [surface.state.save]

1 A surface object provides an interface to save its current state and subsequently restore it.

2 Save and restore operations are performed using surface::save and surface::restore, respectively.

3 Save and restore operations may be nested.

4 Each call to surface::restore restores a surface object’s state to its values at the time of the most recent call to surface::save.

§ 13.10.3.2 145
The following list denotes each item of observable state that shall be saved by `surface::save` and restored by `surface::restore`, using the state item names listed in Table 20:

(5.1) — Current Brush
(5.2) — General Antialiasing
(5.3) — Dash Pattern
(5.4) — Fill Rule
(5.5) — Line Cap
(5.6) — Line Join
(5.7) — Line Width
(5.8) — Miter Limit
(5.9) — Compositing Operator
(5.10) — Tolerance
(5.11) — Clip Area
(5.12) — Current Path
(5.13) — Immediate Path
(5.14) — Transformation Matrix
(5.15) — Font Face
(5.16) — Font Matrix
(5.17) — Font Options

13.10.4 surface rendering and composing operations

The `surface` class provides five fundamental rendering and composing operations:

Table 21 — `surface` rendering and composing operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Painting</td>
<td><code>surface::paint</code></td>
</tr>
<tr>
<td>Filling</td>
<td><code>surface::fill, surface::fill_immediate</code></td>
</tr>
<tr>
<td>Stroking</td>
<td><code>surface::stroke, surface::stroke_immediate</code></td>
</tr>
<tr>
<td>Masking</td>
<td><code>surface::mask, surface::mask_immediate</code></td>
</tr>
<tr>
<td>Typesetting</td>
<td><code>surface::draw_text</code></td>
</tr>
</tbody>
</table>

The filling, stroking, and masking operations each provide two functions, not including overloads, for invoking their functionality.

Certain rendering and composing operations require a `source path`, which is a path geometry graphics resource.

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.
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 geometry graphics resource formed in the manner specified by 10.1.2 from the (const reference to) `vector<path_data_item>` returned by calling the `path_factory::data_ref` member function on the surface's Immediate Path.

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` object from the result of calling `surface::immediate` on `s` and then immediately setting that `path` object as the Current Path by calling `surface::path` on `s` using that `path` 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` object from the result of calling `surface::immediate` on `s` and then immediately setting that `path` object as the Current Path by calling `surface::path` on `s` using that `path` 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` object from the result of calling `surface::immediate` on `s` and then immediately setting that `path` object as the Current Path by calling `surface::path` on `s` using that `path` object as the argument to that function.

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.

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]

Each of the five 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 one minimum argument: an object, used as the mask (see 13.10.7.4), that is of type `brush` or type `surface`.
5. The typesetting operation has two minimum arguments: an object, used as the text that is rendered and composed to the surface (see 13.10.7.5), that is of type `string`; and, an object, used to specify the coordinates at which the origin of the first character of the text is positioned (see 13.10.7.5), of type vector_2d.

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.

For the masking operation, where a `surface` object is used as the mask argument, it may be immediately followed by an argument of type `vector_2d` which shall be used as the mask surface origin (see 13.10.7.4). If a mask surface origin argument is passed to the masking operation, it shall be considered part of the set of minimum arguments for that operation.
A source brush is composed of a pixmap, an extend value, a filter value, and a matrix_2d object.

[Note: A brush object provides all of the components of a source brush. As such, brush objects will commonly be used as source brushes. — end note]

The rendering and composing operations shall be provided with a source brush each time they are invoked.

A source brush may be provided by a brush object or by an identified set of arguments which, together, provide all of the components of a source brush.

When a rendering and composing operation is invoked with only those arguments that are part of its set of minimum arguments, the surface’s Current Brush shall be the source brush.

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):

(17.1) — Where there is one extra argument and it is a brush object, the source brush shall be that brush object.

(17.2) — 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.

(17.3) — 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 with 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 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::extend was called on b with the extend object extra argument passed to brush::extend 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. If there is an extra argument of type vector_2d, the source brush shall produce the same observable results that b would produce after brush::matrix was called on b with the result of passing the vector_2d object extra argument to matrix_2d::init_translate passed to brush::matrix as an argument.

5. If there is an extra argument of type matrix_2d, 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.

[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]

When a rendering and composing operation needs to obtain a value for a pixel from its source brush, it shall do so as specified in 13.10.5.

A surface object shall not be used as a mask or as a source brush for any rendering and composing operations invoked on that surface object. No diagnostic is required. [Note: When a brush object b is created from a surface object s using the surface_brush_factory class, b may be used as a mask or as a source brush for rendering and composing operations invoked on s because the surface_brush_factory class creates a
13.10.5 Sampling from a source brush

This section is forthcoming in a future revision.

13.10.6 Effects of its filter value of a brush object

The effects of the filter value on a brush object are documented in 12.3.

The initial filter value of a brush object depends on the brush factory used to create it and is documented in the brush constructors for each type of brush factory.

13.10.7 Effects of its matrix_2d value on a brush object

The initial matrix_2d value of a brush object is matrix_2d::init_identity().

If the brush's brush_type is brush_type::solid_color, the brush is treated as if its data area is an infinite plane. As such the brush's extend, filter, and matrix_2d values do not matter. Regardless of the coordinate from which data from the brush is requested, the brush will always return the color which the solid_color brush factory that was used to create it had at the time the brush was created.

All brushes except brush_type::solid_color brushes are sensitive to coordinates and as such will need to have an appropriate matrix_2d value to achieve the desired outcome.

[Note: Sometimes the initial matrix_2d value will be the appropriate value. As such there is no requirement to set a matrix_2d value unless you need something other than the initial value. — end note]

When it is necessary for the implementation to retrieve data from a brush object to calculate the resulting color value of a coordinate on a surface, the coordinate is first transformed using the brush object's matrix_2d.

A brush object's matrix_2d value transforms a coordinate from the surface's coordinate space to the brush's coordinate space.

[Example: Take a brush b which has a matrix_2d value of matrix_2d::init_scale({ 2.0, 0.25 }).]

If a request for data from b at the point vector_2d(30.0, 18.0) is made, by operation of b's matrix_2d, the actual data that will be retrieved is the data at vector_2d(60.0, 4.5) (after application of b's extend value as described above).

Notice that although the x axis scale is set to 2.0 and the y axis scale is set to 0.25, the actual effect is the same as if the b's data itself had been scaled to the inverse of those values. This is because the transformation goes from the surface's coordinate space to the brush's coordinate space rather than from the brush's coordinate space to the surface's coordinate space. — end example]

[Note: For the same reason as explained in the example above, if you wish to translate the origin point of a brush to the point vector_2d p of a surface, you would do so by setting matrix_2d::init_translate(-p) as the brush's matrix_2d value. — end note]

13.10.7.1 surface painting

When a Painting operation is initiated on a surface, the implementation shall compose the source brush with the underlying graphics data graphics resource in the manner specified by the surface's current Composition Operator and modify the underlying graphics data graphics resource to reflect the results of that operation.

No part of the underlying graphics data graphics resource that is outside of the Clip Area shall be modified.

13.10.7.2 surface filling

This section is forthcoming in a future revision.

§ 13.10.7.2
13.10.7.3 surface stroking

1 When a Stroking operation is initiated on a surface, the implementation shall carry out the Stroking operation for each path geometry in the source path.

2 After all path geometries have been rendered but before the rendered result is composed to the underlying graphics data graphics resource, the rendered result shall be transformed by the Transformation Matrix. [Example: If an open path geometry 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 axis of 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]

3 The following rules shall apply when a Stroking operation is carried out on a path geometry:

1. No part of the underlying graphics data graphics resource that is outside of the Clip Area shall be modified.

2. If the path geometry 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 geometry is a closed path geometry, 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 be 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 shown then hidden for the length specified by the value. The ends of each shown 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.

13.10.7.4 surface masking

1 This section is forthcoming in a future revision.

13.10.7.5 surface typesetting

1 This section is forthcoming in a future revision.

13.10.8 surface constructors, assignment operators, and destructors

virtual ~surface();

1 Effects: Destroys an object of class surface.
virtual void finish() noexcept;

Effects: Releases all resources managed by the surface object if !s.is_finished(), otherwise does nothing.

Postconditions: s.is_finished() shall return true.

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.

void flush();
void flush(error_code& ec) noexcept;

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.

If the implementation does provide a native handle to the surface object’s underlying graphics data graphics resource, then:

— Any pending rendering and composing operations (13.10.4) 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]

— The implementation may then modify the surface object’s observable state (13.10.3).

— The surface object’s observable state shall then be undefined.

Throws: As specified in Error reporting (3).

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.

Error conditions: The potential errors are implementation-defined.

Implementations should avoid producing errors here.

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.

Notes: Because the surface object’s observable state can change as a result of calling this function, users will typically want to call surface::save 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::restore (assuming a previous, valid call to surface::save). 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;

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.

Throws: As specified in Error reporting (3).

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;

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.

If the implementation does provide a native handle to the surface object’s underlying graphics data graphics resource, then:

— 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.

— 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 \( \text{rectangle\{ round(extents.x()), round( extents.y()), round(extents.width()), round(extents.height())} \). No part of the bounding rectangle shall be outside of the bounds of the underlying graphics data graphics resource; no diagnostic is required.

Throws: As specified in Error reporting (3).

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.

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 ]

Error conditions: The errors, if any, produced by this function are implementation-defined.

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);

Effects: Creates a mapped_surface object and calls action using it.

The mapped_surface object is created using *this, which allows direct manipulation of the underlying graphics data graphics resource.

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 \( \text{rectangle\{ round(extents.x()), round( extents.y()), round(extents.width()), round(extents.height())} \). 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.

Throws: As specified in Error reporting (3).

Remarks: Whether changes are committed to the underlying graphics data graphics resource immediately or only when the mapped_surface object is destroyed is unspecified.

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.
**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.

```cpp
virtual void save();
virtual void save(error_code& ec) noexcept;
```

**Effects:** Saves the state specified in 13.10.3.2 as if to an internal stack of saved states.

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

**Error conditions:** `errc::not_enough_memory` if the state cannot be saved.

```cpp
virtual void restore();
virtual void restore(error_code& ec) noexcept;
```

**Effects:** Restores the state of `*this` to the values saved by the most recent call to `surface::save`.

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

**Remarks:** Because this function is only restoring previously saved state, except where the conditions for `io2d_error::invalid_restore` are met, implementations should not generate errors.

**Error conditions:** `io2d_error::invalid_restore` if this function is called without a previous matching call to `surface::save`. Implementations shall not produce `io2d_error::invalid_restore` except under the conditions stated in this paragraph.

Excluding the previously specified error, any errors produced by calling this function are implementation-defined.

```cpp
void brush(nullopt_t) noexcept;
```

**Effects:** Sets the Current Brush (Table 20) to be a `brush` object that is the same as if it was the brush that is the default value of Current Brush 13.10.3.1.

**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]

```cpp
void brush(const experimental::io2d::brush& source);
void brush(const experimental::io2d::brush& source, error_code& ec) noexcept;
```

**Effects:** Sets `source` as the Current Brush (Table 20), 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.

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

**Error conditions:** The errors, if any, produced by this function are implementation-defined.

```cpp
void antialias(experimental::io2d::antialias a) noexcept;
```

**Effects:** Sets General Antialiasing (Table 20) to the value of `a`.

```cpp
void dashes(nullopt_t) noexcept;
```

**Effects:** Sets the Dash Pattern (Table 20) to be the default value of Dash Pattern.
Effects: Sets the Dash Pattern (Table 20) to the value of \( d \).

Throws: As specified in Error reporting (3).

Error conditions: `errc::not_enough_memory` if there is a problem caching the new Dash Pattern.
`io2d_error::invalid_dash` if the new Dash Pattern contains a negative value or if it has values and all of them are \( 0.0 \).

```cpp
void fill_rule(experimental::io2d::fill_rule fr) noexcept;
```

Effects: Sets the Fill Rule (Table 20) to the value of \( fr \).

```cpp
void line_cap(experimental::io2d::line_cap lc) noexcept;
```

Effects: Sets the Line Cap (Table 20) to the value of \( lc \).

```cpp
void line_join(experimental::io2d::line_join lj) noexcept;
```

Effects: Sets the Line Join (Table 20) to the value of \( lj \).

```cpp
void line_width(double width) noexcept;
```

Effects: Sets the Line Width (Table 20) to \( \max(\text{width}, 0.0) \).

```cpp
void miter_limit(double limit) noexcept;
```

Effects: Sets the Miter Limit (Table 20) 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.

Remarks: The implementation-defined minimum value shall not be greater than \( 2.0 \) and the implementation-defined maximum value shall not be less than \( 10.0 \).

Notes: The Miter Limit only applies when the Line Join is set to `line_join::miter`. — end note

```cpp
void compositing_operator(experimental::io2d::compositing_operator co) noexcept;
```

Effects: Sets the Composition Operator to the value of \( co \).

```cpp
void clip(const experimental::io2d::path& p);
void clip(const experimental::io2d::path& p, error_code& ec) noexcept;
```

Effects: Sets the Clip Area (Table 20) 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.

Notes: The Clip Area never increases as a result of this function.

```cpp
void clip_immediate();
void clip_immediate(error_code& ec) noexcept;
```

Effects: Sets the Clip Area (Table 20) to be the intersection of the Clip Area and the Immediate Path (Table 20) where the Immediate Path’s area is determined in the same way as if the Immediate Path were filled according to the Fill Rule.

Notes: The Clip Area never increases as a result of this function.

```cpp
void path(nullopt_t) noexcept;
```
Effects: Set's the Current Path (Table 20).

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

```cpp
void path(const experimental::io2d::path& p);
void path(const experimental::io2d::path& p, error_code& ec) noexcept;
```

Effects: Set's Current Path (Table 20) to p.

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 10.1.2.

### 13.10.10 surface immediate path modifiers

```
experimental::io2d::path_factory& immediate() noexcept;
```

Returns: A reference to the Immediate Path (Table 20).

### 13.10.11 surface render modifiers

```cpp
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 vector_2d& o = vector_2d{ },
          extend e = extend::none, filter f = filter::good);
```

Effects: Performs the Filling rendering and composing operation as specified by 13.10.7.2.

2 The meanings of the parameters are specified by 13.10.4.

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

Error conditions: The errors, if any, produced by this function are implementation-defined.

```cpp
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 vector_2d& o = vector_2d{ },
                    extend e = extend::none, filter f = filter::good);
void fill_immediate(const surface& s, error_code& ec,
                    const vector_2d& o = vector_2d{ }, extend e = extend::none,
                    filter f = filter::good) noexcept;
```
Effects: Performs the Filling rendering and composing operation as specified by 13.10.7.2.

The meanings of the parameters are specified by 13.10.4.

Throws: As specified in Error reporting (3).

Error conditions: The errors, if any, produced by this function are implementation-defined.

Effects: Performs the Painting rendering and composing operation as specified by 13.10.7.1.

The meanings of the parameters are specified by 13.10.4.

Throws: As specified in Error reporting (3).

Error conditions: The errors, if any, produced by this function are implementation-defined.
void stroke(const experimental::io2d::brush& b, error_code& ec) noexcept;
void stroke(const surface& s, const vector_2d& o = vector_2d{},
    extend e = extend::none, filter f = filter::good);
void stroke(const surface& s, error_code& ec,
    const vector_2d& o = vector_2d{},
    extend e = extend::none, filter f = filter::good) noexcept;
void stroke(const surface& s, const matrix_2d& m, extend e = extend::none,
    filter f = filter::good);
void stroke(const surface& s, const matrix_2d& m, error_code& ec,
    extend e = extend::none, filter f = filter::good) noexcept;

Effects: Performs the Stroking rendering and composing operation as specified by 13.10.7.3.
The meanings of the parameters are specified by 13.10.4.

Throws: As specified in Error reporting (3).

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 vector_2d& o = vector_2d{},
    extend e = extend::none, filter f = filter::good);
void stroke_immediate(const surface& s, error_code& ec,
    const vector_2d& o = vector_2d{},
    extend e = extend::none, filter f = filter::good) noexcept;
void stroke_immediate(const surface& s, const matrix_2d& m,
    extend e = extend::none, filter f = filter::good);
void stroke_immediate(const surface& s, const matrix_2d& m, error_code& ec,
    extend e = extend::none, filter f = filter::good) noexcept;

Effects: Performs the Stroking rendering and composing operation as specified by 13.10.7.3.
The meanings of the parameters are specified by 13.10.4.

Throws: As specified in Error reporting (3).

Error conditions: The errors, if any, produced by this function are implementation-defined.

13.10.12 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 vector_2d& o = vector_2d{}, extend e = extend::none,
    filter f = filter::good);
void mask(const experimental::io2d::brush& mb, const surface& s,
error_code& ec, const vector_2d& o = vector_2d{ },
extend e = extend::none, filter f = filter::good) noexcept;
void mask(const experimental::io2d::brush& mb, const surface& s,
const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void mask(const experimental::io2d::brush& mb, const surface& s,
const matrix_2d& m, error_code& ec, extend e = extend::none,
filter f = filter::good) noexcept;
void mask(const surface& ms);
void mask(const surface& ms, error_code& ec) noexcept;
void mask(const surface& ms, const rgba_color& c);
void mask(const surface& ms, const rgba_color& c, error_code& ec) noexcept;
void mask(const surface& ms, const experimental::io2d::brush& b);
void mask(const surface& ms, const experimental::io2d::brush& b,
error_code& ec) noexcept;
void mask(const surface& ms, const surface& s,
const vector_2d& o = vector_2d{ }, extend e = extend::none,
filter f = filter::good);
void mask(const surface& ms, const surface& s, error_code& ec,
const vector_2d& o = vector_2d{ }, extend e = extend::none,
filter f = filter::good) noexcept;
void mask(const surface& ms, const surface& s, const matrix_2d& m,
extend e = extend::none, filter f = filter::good);
void mask(const surface& ms, const surface& s, const matrix_2d& m,
error_code& ec, extend e = extend::none, filter f = filter::good)
noexcept;
void mask(const surface& ms, const surface& s, const vector_2d& mo);
void mask(const surface& ms, const vector_2d& mo, error_code& ec)
noexcept;
void mask(const surface& ms, const vector_2d& mo, const rgba_color& c);
void mask(const surface& ms, const vector_2d& mo, const rgba_color& c,
error_code& ec) noexcept;
void mask(const surface& ms, const vector_2d& mo,
const experimental::io2d::brush& b);
void mask(const surface& ms, const vector_2d& mo,
const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask(const surface& ms, const vector_2d& mo, const surface& s,
const vector_2d& o = vector_2d{ }, extend e = extend::none,
filter f = filter::good);
void mask(const surface& ms, const vector_2d& mo, const surface& s,
const vector_2d& o = vector_2d{ }, extend e = extend::none, filter f = filter::good)
noexcept;
void mask(const surface& ms, const vector_2d& mo, const surface& s,
const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void mask(const surface& ms, const vector_2d& mo, const surface& s,
const matrix_2d& m, error_code& ec, extend e = extend::none,
filter f = filter::good) noexcept;

Effects: Performs the Masking rendering and composing operation as specified by 13.10.7.4.
The meanings of the parameters are specified by 13.10.4.
Throws: As specified in Error reporting (3).
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 vector_2d& o = vector_2d{ }, extend e = extend::none,
    filter f = filter::good);
void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
    error_code& ec, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good) noexcept;
void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
    const matrix_2d& m, extend e = extend::none, filter f = filter::good);
void mask_immediate(const experimental::io2d::brush& mb, const surface& s,
    const matrix_2d& m, error_code& ec, extend e = extend::none,
    filter f = filter::good) noexcept;
void mask_immediate(const surface& ms);
void mask_immediate(const surface& ms, error_code& ec) noexcept;
void mask_immediate(const surface& ms, const rgba_color& c);
void mask_immediate(const surface& ms, const rgba_color& c, error_code& ec)
    noexcept;
void mask_immediate(const surface& ms, const experimental::io2d::brush& b);
void mask_immediate(const surface& ms, const experimental::io2d::brush& b,
    error_code& ec) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo);
void mask_immediate(const surface& ms, const vector_2d& mo, error_code& ec)
    noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const rgba_color& c);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const rgba_color& c, error_code& ec) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const experimental::io2d::brush& b);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const experimental::io2d::brush& b, error_code& ec) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const surface& s, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good);
void mask_immediate(const surface& ms, const vector_2d& mo,
    const surface& s, error_code& ec, const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good) noexcept;
void mask_immediate(const surface& ms, const vector_2d& mo,
    const vector_2d& o = vector_2d{ },
    extend e = extend::none, filter f = filter::good) noexcept;

§ 13.10.12
const surface& s, const matrix_2d& m, extend e = extend::none,
filter f = filter::good);
void mask_immediate(const surface& ms, const vector_2d& mo,
const surface& s, const matrix_2d& m, error_code& ec,
extend e = extend::none, filter f = filter::good) noexcept;

Effects: Performs the Masking rendering and composing operation as specified by 13.10.7.4.
The meanings of the parameters are specified by 13.10.4.
Throws: As specified in Error reporting (3).
Error conditions: The errors, if any, produced by this function are implementation-defined.

13.10.13 surface text render modifiers

Effects: Performs the Typesetting rendering and composing operation as specified by 13.10.7.5.
The meanings of the parameters are specified by 13.10.4.
Throws: As specified in Error reporting (3).
Error conditions: The errors, if any, produced by this function are implementation-defined.

13.10.14 surface transformation modifiers

Effects: Sets Transformation Matrix (Table 20) to the value of m.
Throws: As specified in Error reporting (3).
Error conditions: io2d_error::invalid_matrix if calling !m.is_invertible().

13.10.15 surface font modifiers
void font_face(const string& typeface, font_slant sl, font_weight w);
void font_face(const string& typeface, font_slant sl, font_weight w, 
              error_code& ec) noexcept;

1. Effects: Sets Font Face (Table 20) to the result of constructing a simple_font_face object using the arguments to this function.
2. If an error occurs in constructing the simple_font_face object, Font Face shall retain the value it had prior to the execution of this function and that error shall be propagated back.
3. Throws: As specified in Error reporting (3).
4. Error conditions: The errors, if any, produced by this function are implementation-defined.

void font_face(const experimental::io2d::font_face& f);
void font_face(const experimental::io2d::font_face& f, error_code& ec) noexcept;

5. Effects: Sets Font Face (Table 20) to f.
6. Throws: As specified in Error reporting (3).
7. Error conditions: The errors, if any, produced by this function are implementation-defined.

void font_size(double s);
void font_size(double s, error_code& ec) noexcept;

8. Effects: Sets Font Matrix (Table 20) to the value of matrix_2d::init_scale{ { s, s } }.
9. Throws: As specified in Error reporting (3).
10. Error conditions: errc::invalid_argument if s <= 0.0.

void font_matrix(const matrix_2d& m);
void font_matrix(const matrix_2d& m, error_code& ec) noexcept;

11. Effects: Sets Font Matrix (Table 20) to the value of m.
12. Throws: As specified in Error reporting (3).
13. Error conditions: io2d_error::invalid_matrix if calling !m.is_invertible().

void font_options(const font_options& fo) noexcept;

14. Effects: Sets Font Options (Table 20) to fo.

13.10.16 surface state observers

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 (13.2.3) that appropriately describes the underlying graphics data graphics resource.

experimental::io2d::brush brush() const noexcept;
3. Returns: The Current Brush (Table 20).

experimental::io2d::antialias antialias() const noexcept;
4. Returns: The value of General Antialiasing (Table 20).
experimental::io2d::dashes dashes() const;
experimental::io2d::dashes dashes(error_code& ec) const noexcept;

5 Returns: The Dash Pattern (Table 20).

6 Throws: As specified in Error reporting (3).

7 Remarks: See 13.10.7.3 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 20).

experimental::io2d::line_cap line_cap() const noexcept;

10 Returns: The value of Line Cap (Table 20).

experimental::io2d::line_join line_join() const noexcept;

11 Returns: The value of Line Join (Table 20).

double line_width() const noexcept;

12 Returns: The value of Line Width (Table 20).

double miter_limit() const noexcept;

13 Returns: The value of Miter Limit (Table 20).

experimental::io2d::compositing_operator compositing_operator() const noexcept;

14 Returns: The value of Composition Operator (Table 20).

rectangle clip_extents() const noexcept;

15 Returns: A rectangle that specifies the smallest bounding box which contains the Clip Area (Table 20).

bool in_clip(const vector_2d& pt) const noexcept;

16 Returns: If the point pt is outside of the Clip Area (Table 20), 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 20).

If an error_code& argument is passed and the error io2d_error::clip_not_representable occurs, returns an empty vector<rectangle> object.

18 Throws: As specified in Error reporting (3).

19 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]

20 errc::not_enough_memory if a failure to allocate memory occurs.

§ 13.10.16
13.10.17  surface render observers

rectangle fill_extents() const noexcept;

Returns: A rectangle that specifies the smallest bounding box which contains all areas in which a Filling operation (13.10.7.2) can have an effect using the Current Path (Table 20) 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.

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.

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;

Returns: A rectangle that specifies the smallest bounding box which contains all areas in which a Filling operation (13.10.7.2) can have an effect using the Immediate Path (Table 20) 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.

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.

Throws: As specified in Error reporting (3).

Remarks: This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 10.1.2.

Error conditions: io2d_error::no_current_point if, when processing the Immediate Path’s path geometries, an operation was encountered which required a current point and the current path geometry had no current point.

io2d_error::invalid_matrix if, when processing the Immediate Path’s path geometries, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

Other errors, if any, produced by this function are implementation-defined.

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;

Returns: If the point pt is not within any of the areas in which a Filling operation (13.10.7.2) can have an effect using the Current Path (Table 20) 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.

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 \texttt{surface::user_to_surface} prior to determining whether or not it is within any of the Filling operation areas.

\textbf{Notes:} The result does not mean that the content at the point \texttt{pt} would be changed if \texttt{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 \texttt{surface::fill}.

\begin{verbatim}
bool in_fill_immediate(const vector_2d& pt) const;
bool in_fill_immediate(const vector_2d& pt, error_code& ec) const noexcept;
\end{verbatim}

\begin{verbatim}
Returns: If the point \texttt{pt} is not within any of the areas in which a Filling operation (13.10.7.2) can have an effect using the Immediate Path (Table 20) and the value of Fill Rule, this function returns \texttt{false}; otherwise it returns \texttt{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.

For purposes of calculating the return value, the processed coordinates of the Immediate Path shall be transformed using \texttt{surface::user_to_surface} before any other calculations are performed. The value of \texttt{pt} shall also be transformed using \texttt{surface::user_to_surface} prior to determining whether or not it is within any of the Filling operation areas.

\textbf{Throws:} As specified in Error reporting (3).
\end{verbatim}

\textbf{Remarks:} This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 10.1.2.

\textbf{Error conditions:} \texttt{io2d_error::no_current_point} if, when processing the Immediate Path’s path geometries, an operation was encountered which required a current point and the current path geometry had no current point.

\texttt{io2d_error::invalid_matrix} if, when processing the Immediate Path’s path geometries, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

Other errors, if any, produced by this function are implementation-defined.

\textbf{Notes:} The result does not mean that the point \texttt{pt} would be changed if \texttt{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 \texttt{surface::fill}.

\begin{verbatim}
rectangle stroke_extents() const noexcept;
\end{verbatim}

\begin{verbatim}
Returns: A \texttt{rectangle} that specifies the smallest bounding box which contains all areas in which a Stroking operation (13.10.7.3) can have an effect using the Current Path (Table 20), 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.

For purposes of calculating the return value, the coordinates of the Current Path shall be transformed using \texttt{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 \texttt{surface::surface_to_user} and the \texttt{rectangle} that is returned shall be composed of those transformed values.

\textbf{Notes:} The resulting bounding box is not the same as the area that would be changed if \texttt{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 \texttt{surface::stroke}.

\section*{13.10.17}
Returns: A rectangle that specifies the smallest bounding box which contains all areas in which a Stroking operation (13.10.7.3) can have an effect using the Current Path (Table 20), 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.

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.

Throws: As specified in Error reporting (3).

Remarks: This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 10.1.2.

Error conditions: `io2d_error::no_current_point` if, when processing the Immediate Path’s path geometries, an operation was encountered which required a current point and the current path geometry had no current point.

`io2d_error::invalid_matrix` if, when processing the Immediate Path’s path geometries, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

Other errors, if any, produced by this function are implementation-defined.

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`.

Returns: If the point pt is not within any of the areas in which a Stroking operation (13.10.7.3) can have an effect using the Current Path (Table 20), 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.

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.

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`.

Returns: If the point pt is not within any of the areas in which a Stroking operation (13.10.7.3) can have an effect using the Immediate Path (Table 20), 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.

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.

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

**Remarks:** This function requires that the Immediate Path be processed into a usable form as if in the manner specified in 10.1.2.

**Error conditions:**
- `io2d_error::no_current_point` if, when processing the Immediate Path’s path geometries, an operation was encountered which required a current point and the current path geometry had no current point.
- `io2d_error::invalid_matrix` if, when processing the Immediate Path’s path geometries, an operation was encountered which required the current transformation matrix to be invertible and the matrix was not invertible.

Other errors, if any, produced by this function are implementation-defined.

**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`.

```
experimental::io2d::font_extents font_extents() const noexcept;
```

**Returns:** A `font_extents` object which specifies metrics for Font Face (Table 20). See the specification of the `font_extents` class (11.7) for more information.

The Transformation Matrix is disregarded for purposes of calculating the metrics.

```
experimental::io2d::text_extents text_extents(const string& utf8) const;
experimental::io2d::text_extents text_extents(const string& utf8, error_code& ec) const noexcept;
```

**Returns:** A `text_extents` object which specifies metrics for `utf8` if it was rendered using Font Face (Table 20). See the specification of the `text_extents` class (11.8) for more information.

The Transformation Matrix is disregarded for purposes of calculating the metrics.

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

**Error conditions:**
- `errc::invalid_argument` if `utf8` is not a valid UTF-8 string.

Other errors, if any, produced by this function are implementation-defined.

### 13.10.18 surface transformation observers

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

**Returns:** The Transformation Matrix (Table 20).

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

**Returns:** The result of calling `matrix_2d::transform_point` on Transformation Matrix (Table 20) with `pt` as the argument to that function.

```
vector_2d user_to_surface_distance(const vector_2d& dpt) const noexcept;
```
Returns: The result of calling \texttt{matrix\_2d::transform\_distance} on Transformation Matrix (Table 20) with \texttt{dpt} as the argument to that function.

\begin{verbatim}
vector\_2d surface\_to\_user(const vector\_2d& pt) const noexcept;
\end{verbatim}

\textbf{Returns:} The result of creating a copy of Transformation Matrix (Table 20), calling \texttt{matrix\_2d::invert} on that copy, and then calling \texttt{matrix\_2d::transform\_point} on the copy with \texttt{pt} as the argument to that function.

\begin{verbatim}
vector\_2d surface\_to\_user\_distance(const vector\_2d& dpt) const noexcept;
\end{verbatim}

\textbf{Returns:} The result of creating a copy of Transformation Matrix (Table 20), calling \texttt{matrix\_2d::invert} on that copy, and then calling \texttt{matrix\_2d::transform\_distance} on the copy with \texttt{dpt} as the argument to that function.

13.10.19 \textbf{surface font observers} [surface.observers.font]

\begin{verbatim}
matrix\_2d font\_matrix() const noexcept;
\end{verbatim}

\textbf{Returns:} The Font Matrix (Table 20).

\begin{verbatim}
experimental::io2d::font\_options font\_options() const noexcept;
\end{verbatim}

\textbf{Returns:} The Font Options (Table 20).

\begin{verbatim}
experimental::io2d::font\_face font\_face() const;
experimental::io2d::font\_face font\_face(error\_code& ec) const noexcept;
\end{verbatim}

\textbf{Returns:} The Font Face (Table 20). \textbf{Throws:} As specified in Error reporting (3).

\begin{verbatim}
error\_code& ec) const noexcept;
\end{verbatim}

\textbf{Error conditions:} The errors, if any, produced by this function are implementation-defined.

13.11 \textbf{Class image\_surface} [imagesurface]

13.11.1 \textbf{image\_surface synopsis} [imagesurface.synopsis]

\begin{verbatim}
namespace std { namespace experimental { namespace io2d { inline namespace v1 {

class image\_surface : public surface {
public:
  // 13.11.3, 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();

  // 13.11.4, 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;

\end{verbatim}

§ 13.11.1
13.11.2 image_surface Description

The class `image_surface` derives from the `surface` class and provides an interface to a raster graphics data graphics resource.

[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]

13.11.3 image_surface constructors and assignment operators

```cpp
image_surface(experimental::io2d::format fmt, int width, int height);
image_surface(experimental::io2d::format fmt, int width, int height, error_code& ec) noexcept;
```

**Effects:** Constructs an object of type `image_surface`.

**Postconditions:** `this->format() == fmt`.

**Postconditions:** `this->width() == width`.

**Postconditions:** `this->height() == height`.

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

**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]

**Error conditions:** The errors, if any, produced by this function are implementation-defined.

```cpp
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;
```

**Effects:** Constructs an object of type `image_surface`.

**Postconditions:** `this->format() == fmt`.

**Postconditions:** `this->width() == width`.

**Postconditions:** `this->height() == height`.

**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

Throws: As specified in Error reporting (3).

Error conditions: `io2d_error::invalid_stride` if `format_stride_for_width(fmt, width) * height != data.size()`.

Other errors, if any, produced by this function are implementation-defined.

`virtual ~image_surface();`

Effects: Destroys an object of type `image_surface`.

### 13.11.4 image_surface modifiers

```c
void data(const vector<unsigned char>& data);
void data(const vector<unsigned char>& data, error_code& ec) noexcept;
```

Effects: Any pending rendering and composing operations (13.10.4) shall be performed.

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]

Throws: As specified in Error reporting (3).

Error conditions: `io2d_error::invalid_stride` if `format_stride_for_width(fmt, width) * height != data.size()`.

Other errors, if any, produced by this function are implementation-defined.

```c
vector<unsigned char> data();
vector<unsigned char> data(error_code& ec) noexcept;
```

Effects: Any pending rendering and composing operations (13.10.4) shall be performed.

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.

Throws: As specified in Error reporting (3).

Error conditions: `errc::not_enough_memory` if there was a failure to allocate memory.

Notes: This would normally be an observer function but the requirement that "[a]ny pending rendering and composing operations (13.10.4) 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.

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.
13.11.5 image_surface observers

experimental::io2d::format format() const noexcept;

Returns: The pixel format of the image_surface object.

Remarks: If the image_surface object is invalid, this function shall return experimental::io2d::format::invalid.

int width() const noexcept;

Returns: The number of pixels per horizontal line of the image_surface object.

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;

Returns: The number of horizontal lines of pixels in the image_surface object.

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;

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]

Remarks: This function shall return the value 0 if this->format() == experimental::io2d::format::unknown || this->format() == experimental::io2d::format::invalid.

13.12 Class display_surface

13.12.1 display_surface synopsis

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

class display_surface : public surface {
public:
  // 13.12.4, 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);

display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat, error_code& ec,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox)
  noexcept;

display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  int preferredDisplayWidth, int preferredDisplayHeight,
  experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox);

display_surface(int preferredWidth, int preferredHeight,
  experimental::io2d::format preferredFormat,
  int preferredDisplayWidth, int preferredDisplayHeight, error_code& ec,}
virtual ~display_surface();

// 13.12.5, modifiers:
virtual void save() override;
virtual void save(error_code& ec) noexcept override;
virtual void restore() override;
virtual void restore(error_code& ec) noexcept override;

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::nullopt_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;
int show();
int show(error_code& ec);
void exit_show(int milliseconds);
void exit_show(int milliseconds, error_code& ec);

// 13.12.6, 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;
```cpp
bool auto_clear() const noexcept;
};
} } } }

13.12.2 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 22) is called in a continous loop in order to dynamically 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 to the Display Buffer. The Display Buffer is then shown to the user via the output device.

5 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.

6 Implementations need not support the simultaneous existence of multiple display_surface objects.

7 All functions inherited from surface that operate on its underlying graphics data graphics resource shall operate on the Back Buffer.

13.12.3 display_surface state
[displaysurface.state]

1 Table Table 22 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 13.10.3.1.

Table 22 — Display surface observable state

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letterbox</td>
<td>brush</td>
<td>This is the brush that shall be used as specified by</td>
<td>brush{ rgba_color::black() } }</td>
</tr>
<tr>
<td>Brush</td>
<td></td>
<td>scaling::letterbox (Table 19)</td>
<td></td>
</tr>
<tr>
<td>Scaling</td>
<td>scaling</td>
<td>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</td>
<td>antialias::default_-antialias</td>
</tr>
</tbody>
</table>
Table 22 — Display surface observable state (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw Width</td>
<td>int</td>
<td>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.</td>
<td>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]</td>
</tr>
</tbody>
</table>

§ 13.12.3 173
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
<th>Note: It is impossible to create a <code>display_surface</code> object without providing a preferred Draw Height value; as such a default value cannot exist. — end note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw Height</td>
<td><code>int</code></td>
<td>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.</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Draw Format</td>
<td><code>experimental::io2d::format</code></td>
<td>The pixel format of the Back Buffer. When a <code>display_surface</code> 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</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
Table 22 — Display surface observable state (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Width</td>
<td>int</td>
<td>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</td>
<td>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]</td>
</tr>
</tbody>
</table>

§ 13.12.3
Table 22 — Display surface observable state (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Height</td>
<td>int</td>
<td>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.</td>
<td>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]</td>
</tr>
<tr>
<td>Draw Callback</td>
<td>function&lt;void(display_surface&amp;)&gt;</td>
<td>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.</td>
<td>nullptr</td>
</tr>
</tbody>
</table>
Table 22 — Display surface observable state (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size Change</strong></td>
<td></td>
<td>function&lt;void(display_surface&amp;)&gt;</td>
<td>nullptr</td>
</tr>
<tr>
<td>Callback</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>function&lt;void(display_surface&amp;)&gt;</td>
<td>nullptr</td>
</tr>
<tr>
<td><strong>User Scaling</strong></td>
<td></td>
<td>function&lt;experimental::io2d::rectangle(const display_surface&amp;, bool&amp;)&gt;</td>
<td></td>
</tr>
<tr>
<td>Callback</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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]
Table 22 — Display surface observable state (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Clear</td>
<td>bool</td>
<td>If true the implementation shall call <code>surface::clear</code>, which shall clear the Back Buffer, immediately before it is required to execute the Draw Callback.</td>
</tr>
</tbody>
</table>

13.12.4 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);
display_surface(int preferredWidth, int preferredHeight, experimental::io2d::format preferredFormat, error_code& ec, experimental::io2d::scaling scl = experimental::io2d::scaling::letterbox) noexcept;

Effects: Constructs an object of type `display_surface`.

1 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.

2 The preferredFormat parameter specifies the preferred pixel format value for Draw Format.

3 The value of Scaling Type shall be the value of `scl`.

4 All other observable state data shall have their default values.

5 Throws: As specified in Error reporting (3).

6 Error conditions: `errc::invalid_argument` if `preferredWidth <= 0`, `preferredHeight <= 0`, or `preferredFormat == experimental::io2d::format::invalid`.

7 `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.

8 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);
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) noexcept;

Effects: Constructs an object of type `display_surface`.

9 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.
The preferredFormat parameter specifies the preferred pixel format value for Draw Format.

The value of Scaling Type shall be the value of scl.

All other observable state data shall have their default values.

Throws: As specified in Error reporting (3).

Error conditions: errc::invalid_argument if preferredWidth <= 0, preferredHeight <= 0, preferredDisplayWidth <= 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.

Other errors, if any, produced by this function are implementation-defined.

### 13.12.5 display_surface modifiers

virtual void save() override;
virtual void save(error_code& ec) noexcept override;

Effects: Calls surface::save.

Implementations may save additional data provided that it does not alter the observable state of the display_surface object.

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if the state cannot be saved.

virtual void restore() override;
virtual void restore(error_code& ec) noexcept override;

Effects: Calls surface::restore.

If the implementation saved additional data as per display_surface::save, it should restore that data. Otherwise it shall discard that data.

Throws: As specified in Error reporting (3).

Remarks: Because this function is only restoring previously saved state, except where the conditions for io2d_error::invalid_restore are met, implementations should not generate errors.

Error conditions: io2d_error::invalid_restore if this function is called without a previous matching call to surface::save. Implementations shall not produce io2d_error::invalid_restore except under the conditions stated in this paragraph.

Excluding the previously specified error, any errors produced by calling this function are implementation-defined.

void draw_callback(const function<void(display_surface& sfc)>& fn) noexcept;

Effects: Sets the Draw Callback to fn.

void size_change_callback(const function<void(display_surface& sfc)>& fn) noexcept;

Effects: Sets the Size Change Callback to fn.

void width(int w);
void width(int w, error_code& ec) noexcept;
**Effects:** If the value of Draw Width is the same as \( w \), this function does nothing.

Otherwise, Draw Width is set as specified by Table 22 with \( w \) treated as being the preferred Draw Width.

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. In the event of failure, Draw Width shall retain its previous value.

\[ \text{Note: The intent of the previous paragraph is to ensure that, no matter the result, a valid Back Buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new Back Buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing Back Buffer even if they cannot determine in advance that creating the new Back Buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a Back Buffer with the previous dimensions. Regardless, there must be a valid Back Buffer when this call completes. —end note}\]

The value of the Back Buffer’s pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

If the attempt to create a new Back Buffer with the updated dimensions fails, implementations shall attempt to recreate the Back Buffer with its old dimensions.

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

**Error conditions:**
- `errc::invalid_argument` if \( w \leq 0 \) 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.

```cpp
void height(int h);
void height(int h, error_code& ec) noexcept;
```

**Effects:** If the value of Draw Height is the same as \( h \), this function does nothing.

Otherwise, Draw Height is set as specified by Table 22 with \( h \) treated as being the preferred Draw Height.

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. In the event of failure, Draw Height shall retain its previous value.

\[ \text{Note: The intent of the previous paragraph is to ensure that, no matter the result, a valid Back Buffer continues to exist. Sometimes implementations will be able to determine that the new dimensions are valid but that to create the new Back Buffer successfully the previous one must be destroyed. The previous paragraph gives implementors that leeway. It goes even further in that it allows implementations to destroy the existing Back Buffer even if they cannot determine in advance that creating the new Back Buffer will succeed, provided that they can guarantee that if the attempt fails they can always successfully recreate a Back Buffer with the previous dimensions. Regardless, there must be a valid Back Buffer when this call completes. —end note}\]
The value of the Back Buffer’s pixel data shall be unspecified upon completion of this function regardless of whether it succeeded.

If the attempt to create a new Back Buffer with the updated dimensions fails, implementations shall attempt to recreate the Back Buffer with its old dimensions.

Throws: As specified in Error reporting (3).

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.

```cpp
void display_width(int w);
void display_width(int w, error_code& ec) noexcept;
```

Effects:
Postconditions:
Throws: As specified in Error reporting (3).

Error conditions:

```cpp
void display_height(int h);
void display_height(int h, error_code& ec) noexcept;
```

Effects:
Postconditions:
Throws: As specified in Error reporting (3).

Error conditions:

```cpp
void dimensions(int w, int h);
void dimensions(int w, int h, error_code& ec) noexcept;
```

Effects:
Postconditions:
Throws: As specified in Error reporting (3).

Error conditions:

```cpp
void display_dimensions(int dw, int dh);
void display_dimensions(int dw, int dh, error_code& ec) noexcept;
```

Effects:
Postconditions:
Throws: As specified in Error reporting (3).

Error conditions:

```cpp
void scaling(experimental::io2d::scaling scl) noexcept;
```

Effects: Sets Scaling Type to the value of scl.

```cpp
void user_scaling_callback(const function<experimental::io2d::rectangle( const display_surface&, bool&)>& fn) noexcept;
```
Effects: Sets the User Scaling Callback to \texttt{fn}.

\begin{verbatim}
void letterbox_brush(experimental::nullopt_t) noexcept;
\end{verbatim}

Effects: Sets the Letterbox Brush to its default value.

\begin{verbatim}
void letterbox_brush(const rgba_color& c);
void letterbox_brush(const rgba_color& c, error_code& ec) noexcept;
\end{verbatim}

Effects: Sets the Letterbox Brush to a value as if \texttt{experimental::io2d::brush\{ solid_color\_brush\_factory\{ \} \} }.

\textit{Throws}: As specified in Error reporting (3).

\textit{Error conditions}: The errors, if any, produced by this function are implementation-defined.

\begin{verbatim}
void letterbox_brush(const experimental::io2d::brush& b);
void letterbox_brush(const experimental::io2d::brush& b, error_code& ec) noexcept;
\end{verbatim}

Effects: Sets the Letterbox Brush to \texttt{b}.

\textit{Throws}: As specified in Error reporting (3).

\textit{Error conditions}: The errors, if any, produced by this function are implementation-defined.

\begin{verbatim}
void auto_clear(bool val) noexcept;
\end{verbatim}

Effects: Sets Auto Clear to the value of \texttt{val}.

\begin{verbatim}
int show();
int show(error_code& ec);
\end{verbatim}

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. Evaluate Auto Clear and perform the actions required by its specification, if any.

4. Run the Draw Callback.

5. Ensure that all operations from the Draw Callback that can effect the Back Buffer have completed.

6. Transfer the contents of the Back Buffer to the Display Buffer using sampling. 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.

If \texttt{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.

No later than when this function returns, the output device shall cease to display the contents of the Display Buffer.

What the output device shall display when it is not displaying the contents of the Display Buffer is unspecified.

\textit{Returns}: The possible values and meanings of the possible values returned are implementation-defined.
Throws: As specified in Error reporting (3).

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.

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.

Other errors, if any, produced by this function are implementation-defined.

```cpp
void exit_show(int ms) noexcept;
```

Requires: This function shall only be called from the Draw Callback; no diagnostic is required.

Effects: The implementation shall initiate the process of exiting the `display_surface::show` function's continuous loop.

Implementations shall not wait until the `display_surface::show` function’s continuous loop ends before returning from this function.

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.

A termination time duration shall then be determined as follows:

1. If the value `ms` is negative, the termination time duration shall be an unspecified number of milliseconds.
2. Otherwise the termination time duration shall be `ms` milliseconds.

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.

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.

### 13.12.6 display_surface observers

```cpp
experimental::io2d::format format() const noexcept;
```

Returns: The value of Draw Format.

```cpp
int width() const noexcept;
```

Returns: The Draw Width.

```cpp
int height() const noexcept;
```

Returns: The Draw Height.

```cpp
int display_width() const noexcept;
```

Returns: The Display Width.

```cpp
int display_height() const noexcept;
```

Returns: The Display Height.

```cpp
tuple<int, int> dimensions() const noexcept;
```

§ 13.12.6
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;

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;

Returns: The Scaling Type.

function<experimental::io2d::rectangle(const display_surface&, bool&)> user_scaling_callback() const noexcept;

function<experimental::io2d::rectangle(const display_surface&, bool&)> user_scaling_callback(error_code& ec) const noexcept;

Returns: A copy of User Scaling Callback.

Throws: As specified in Error reporting (3).

Error conditions: errc::not_enough_memory if a failure to allocate memory occurs.

experimental::io2d::brush letterbox_brush() const noexcept;

Returns: The Letterbox Brush.

bool auto_clear() const noexcept;

Returns: The value of Auto Clear.

13.13 Class mapped_surface

13.13.1 mapped_surface synopsis

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

class mapped_surface {

public:

// 13.13.3, 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();

// 13.13.4, 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;

// 13.13.5, 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;

§ 13.13.1 184
13.13.2 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 (13.7).

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.

13.13.3 mapped_surface constructors and assignment operators [mappedsurface.cons]

~mapped_surface();

Effects: Destroys an object of type mapped_surface.

Remarks: Whether any uncommitted changes are committed during destruction of the mapped_surface object is unspecified.

Uncommitted changes shall not be committed during destruction of the mapped_surface object if doing so would result in an exception.

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.

13.13.4 mapped_surface modifiers [mappedsurface.modifiers]

void commit_changes();
void commit_changes(error_code& ec) noexcept;

Effects: Any uncommitted changes shall be committed.

Throws: As specified in Error reporting (3).

Error conditions: The errors, if any, produced by this function are implementation-defined.

§ 13.13.4
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.

13.13.5 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 \( a + \text{this->stride()} \times h \) shall result in undefined behavior; no diagnostic is required.

\[\text{Error conditions: } \text{io2d_error::null_pointer if this->format() == experimental::io2d::format::unknown} \quad || \quad \text{this->format() == experimental::io2d::format::invalid.}\]

\text{experimental::io2d::format format() const noexcept;}

\textit{Returns:} The pixel format of the mapped surface.

\textit{Remarks:} If the mapped surface is invalid, this function shall return \text{experimental::io2d::format::invalid}.

\text{int width() const noexcept;}

\textit{Returns:} The number of pixels per horizontal line of the mapped surface.

\textit{Remarks:} This function shall return the value 0 if \text{this->format() == experimental::io2d::format::unknown} \quad || \quad \text{this->format() == experimental::io2d::format::invalid.}

\text{int height() const noexcept;}

\textit{Returns:} The number of horizontal lines of pixels in the mapped surface.

\textit{Remarks:} This function shall return the value 0 if \text{this->format() == experimental::io2d::format::unknown} \quad || \quad \text{this->format() == experimental::io2d::format::invalid.}

\text{int stride() const noexcept;}

\textit{Returns:} The length, in bytes, of a horizontal line of the mapped surface. [\textit{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]

\textit{Remarks:} This function shall return the value 0 if \text{this->format() == experimental::io2d::format::unknown} \quad || \quad \text{this->format() == experimental::io2d::format::invalid.}
14 Non-member functions
[nonmemberfunctions]

14.1 Non-member functions synopsis
[nonmemberfunctions.synopsis]

namespace std { namespace experimental { namespace io2d { inline namespace v1 { 
  int format_stride_for_width(format fmt, 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 fmt, int width, int height);
  image_surface make_image_surface(format fmt, int width, int height,
    error_code& ec) noexcept;
} } } } // namespaces std::experimental::io2d::v1

14.2 format_stride_for_width
[nonmemberfunctions.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.

14.3 make_display_surface
[nonmemberfunctions.makedisplaysurface]

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;

1 Returns: Returns a display_surface object that is exactly the same as if the display_surface
constructor was called with the same arguments.
Throws: As specified in Error reporting (3).

Error conditions: The errors, if any, produced by this function are the same as the errors for the equivalent display_surface constructor (13.12.4).

14.4 make_image_surface

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;

Returns: Returns an image_surface object that is exactly the same as if the image_surface constructor was called with the same arguments.

Throws: As specified in Error reporting (3).

Error conditions: The errors, if any, produced by this function are the same as the errors for the equivalent display_surface constructor (13.11.3).
Annex A  (informative)
Bibliography

1 The following is a list of informative resources intended to assist in the understanding or use of this Technical Specification.

Index

2D graphics
   synopsis, 11–13
additive color, 2
aliasing, 2
alpha, 2
anti-aliasing, 2
artifact, 2
aspect ratio, 2

C
   Unicode TR, 1
channel, 1
closed path geometry, 2
color
   transparent black, 126
color model, 2
   RGB, 2
   RGBA, 2
color space, 3
   sRGB, 3
color stop, 6
composing operation, 3
composition algorithm, 3
CSS Colors Specification, 1
cubic Bézier curve, 3
current point, 3
definitions, 1–6
degenerate path geometry, 3
degenerate path segment, 3
filter, 4
final path segment, 4

graphics data, 4
raster, 4
graphics resource, 4
   graphics data graphics resource, 4
   path geometry graphics resource, 4
graphics state data, 5
graphics subsystem, 5

initial path segment, 5

last-move-to point, 5

normalize, 5
open path geometry, 5
path geometry, 5
path segment, 5
pixel, 2
 pixmap, 4
point, 4
premultiplied format, 4

references
   normative, 1
render, 5
rendering operation, 6
sample, 6
scope, 1
standard coordinate space, 1

visual data, 1
visual data format, 1

Bibliography 191
Index of implementation-defined behavior

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

errc::argument_out_of_domain
  what_arg value, 9
errc::invalid_argument
  what_arg value, 9
io2d_error_category
  equivalent, 17
simple_font_face
  typeface, 95
surface
  Font Face, 145
antialias
  subpixel, 120
antialiasing
  best, 121
default, 120
fast, 120
good, 121
color stop
  maximum size, 102
  size_type, 102
Dash Pattern
  offset value, 150
display_surface
  constructor, 178, 179
  height, 181
  letterbox_brush, 182, 183
  restore, 179
  show return value, 182
  unsupported Draw Format, 174
  width, 180
filter
  best, 108
  fast, 108
  good, 108
image_surface
  constructor, 168, 169
  data, 169
io2d_error
  device_error, 16

invalid_status, 15
null_pointer, 15
io2d_error_category
  message, 16
mapped_surface
  commit_changes, 185
Miter Limit
  maximum, 154
  minimum, 154
other error codes
  what_arg value, 9
output device
  what_arg value, 9
presence and meaning of native_handle_type and native_handle, 7
surface
  brush, 153
  fill, 155
  fill_extents_immediate, 163
  fill Immediate, 156
  font _ face, 161, 167
  in_fill_immediate, 164
  in_stroke_immediate, 166
  map, 153
  mark_dirty, 152
  mask, 158
  mask Immediate, 160
  paint, 156
  render_text, 160
  restore, 153
  stroke, 157
  stroke_extents_immediate, 165
  stroke Immediate, 157
  text extents, 166
surface::flush errors, 151

Bibliography

192