

Document Number: P0267R10
Date: 2019-10-07
Revises: P0267R9
Reply to: Michael B. McLaughlin
mikebmcl@gmail.com
Herb Sutter
Microsoft Inc.
hsutter@microsoft.com
Jason Zink
jzink_1@yahoo.com
Guy Davidson
guy@creative-assembly.com
Michael Kazakov
mike.kazakov@gmail.com

Audience:

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 fomattting.

Contents

Contents	ii
List of Tables	v
List of Figures	vi
0 Introduction	1
0.1 Background and rational	1
0.2 Revision history	3
1 Scope	7
2 Normative references	8
3 Terms and definitions	9
4 Error reporting	14
5 Library design overview (Informative)	15
6 Header <experimental/io2d> synopsis	17
7 Colors	24
7.1 Introduction to color	24
7.2 Color usage requirements	24
7.3 Class <code>rgba_color</code>	24
8 Graphics math	34
8.1 General	34
8.2 Requirements	34
9 Graphics surfaces	42
9.1 General	42
9.2 Requirements	42
10 Linear algebra	84
10.1 Class <code>basic_point_2d</code>	84
10.2 Class <code>basic_matrix_2d</code>	87
11 Geometry	92
11.1 Class template <code>basic_display_point</code>	92
11.2 Class <code>basic_bounding_box</code>	93
11.3 Class <code>basic_circle</code>	95
12 Text rendering and display	98
12.1 Intro	98
12.2 Enum class <code>font_size_units</code>	98
12.3 Enum class <code>font_weight</code>	99
12.4 Enum class <code>font_capitalization</code>	99
12.5 Enum class <code>font_stretching</code>	100
12.6 Enum class <code>font_style</code>	100
12.7 Enum class <code>font_line</code>	101
12.8 Enum class <code>font_antialias</code>	101
12.9 Enum class <code>generic_font_names</code>	102
12.10 Class template <code>basic_text_props</code>	102

12.11	Class template <code>basic_font</code>	105
12.12	Class template <code>basic_font_database</code>	108
13	Paths	109
13.1	Overview of paths	109
13.2	Path examples (Informative)	109
13.3	Class template <code>basic_figure_items</code>	113
13.4	Class template <code>basic_interpreted_path</code>	136
13.5	Class <code>basic_path_builder</code>	137
14	Brushes	145
14.1	Overview of brushes	145
14.2	Gradient brushes	145
14.3	Enum class <code>wrap_mode</code>	148
14.4	Enum class <code>filter</code>	149
14.5	Enum class <code>brush_type</code>	150
14.6	Class <code>gradient_stop</code>	150
14.7	Class template <code>basic_brush</code>	151
15	Surface state props	155
15.1	Overview	155
15.2	Enum class <code>antialias</code>	155
15.3	Enum class <code>fill_rule</code>	156
15.4	Enum class <code>line_cap</code>	156
15.5	Enum class <code>line_join</code>	157
15.6	Enum class <code>compositing_op</code>	157
15.7	Enum class <code>format</code>	162
15.8	Enum class <code>scaling</code>	163
15.9	Class template <code>basic_render_props</code>	166
15.10	Class template <code>basic_brush_props</code>	168
15.11	Class template <code>basic_clip_props</code>	169
15.12	Class template <code>basic_stroke_props</code>	172
15.13	Class template <code>basic_fill_props</code>	174
15.14	Class template <code>basic_mask_props</code>	175
15.15	Class template <code>basic_dashes</code>	176
16	Surfaces	179
16.1	Enum class <code>refresh_style</code>	179
16.2	Enum class <code>image_file_format</code>	180
16.3	Overview of surface classes	181
16.4	Class <code>basic_image_surface</code>	190
16.5	Class <code>basic_output_surface</code>	196
16.6	Class template <code>basic_unmanaged_output_surface</code>	203
17	Command lists	209
17.1	Overview of command lists	209
17.2	Class template <code>basic_commands</code>	209
17.3	Class template <code>basic_command_list</code>	233
18	Input	236
19	Standalone functions	237
19.1	Standalone functions synopsis	237
19.2	<code>copy_surface</code>	237
19.3	<code>degrees_to_radians</code>	237
19.4	<code>radians_to_degrees</code>	237
19.5	<code>angle_for_point</code>	237
19.6	<code>point_for_angle</code>	238
19.7	<code>arc_start</code>	238
19.8	<code>arc_center</code>	238

19.9	arc_end	238
A	Bibliography	239
B	Response to P1062R0	240
B.1	Overview of P1062R0	240
B.2	Section 4	240
B.3	Section 5	241
B.4	Section 3	242
B.5	Final thoughts and comments	242
C	Response to P1225R0	243
C.1	Overview of P1225R0	243
C.2	Design	243
C.3	Science and teaching	244
C.4	Abstraction level	244
C.5	Missing details	244
C.6	C++ aesthetics and Conclusion	245
	Index	246
	Index of library names	247
	Index of implementation-defined behavior	255

List of Tables

1	Class identifiers modified since R6	6
2	<code>rgba_color</code> static members values	29
3	Graphics math summary	34
4	X typedef-names	34
5	GraphicsMath type member data	34
6	GraphicsMath requirements	35
7	X:: <code>brushes</code> requirements	44
8	X:: <code>paths</code> typedef-names	46
9	X:: <code>paths</code> type member data	47
10	X:: <code>paths</code> requirements	48
11	X:: <code>surface_state_props</code> typedef-names	54
12	X:: <code>surface_state_props</code> type member data	54
13	X:: <code>surface_state_props</code> requirements	55
14	X:: <code>surfaces</code> typedef-names	60
15	X:: <code>surfaces</code> type member data	60
16	Graphics surfaces requirements	61
17	X:: <code>text</code> typedef-names	78
18	X:: <code>text</code> type member data	78
19	X:: <code>text</code> requirements	79
20	<code>font_size_units</code> enumerator meanings	98
21	<code>font_capitalization</code> enumerator meanings	100
22	<code>font_style</code> enumerator meanings	100
23	<code>font_line</code> enumerator meanings	101
24	<code>font_antialias</code> enumerator meanings	101
25	Path interpretation state data	135
26	Figure item interpretation effects	135
27	<code>wrap_mode</code> enumerator meanings	149
28	<code>filter</code> enumerator meanings	150
29	<code>brush_type</code> enumerator meanings	150
30	<code>antialias</code> enumerator meanings	155
31	<code>fill_rule</code> enumerator meanings	156
32	<code>line_cap</code> enumerator meanings	157
33	<code>line_join</code> enumerator meanings	157
34	<code>compositing_op</code> basic enumerator meanings	159
35	<code>compositing_op</code> blend enumerator meanings	160
36	<code>compositing_op</code> hsl enumerator meanings	162
37	<code>format</code> enumerator meanings	163
38	<code>scaling</code> enumerator meanings	164
39	<code>refresh_style</code> value meanings	179
40	<code>image_file_format</code> enumerator meanings	180
41	surface rendering and compositing operations	181
42	<code>surface</code> rendering and compositing common state data	182
43	surface rendering and compositing specific state data	182
44	Point transformations	183
45	Output surface observable state	186

List of Figures

1	Example 1 result	110
2	Example 2 result	111
3	Path example 3	112
4	Path example 4	113

0 Introduction

[io2d.intro]

0.1 Background and rational

[io2d.overview]

0.1.1 Interactive I/O background

[io2d.background]

- ¹ When C++ was first created, the dominant interactive I/O technology was a console. From its inception, C++ included the ability to interact with users via a console as part of its standard library. Programs that needed to interact with the user would send output information in the form of text to the user and receive input information in the form of text from the user.
- ² The implementation details necessary to provide this functionality varied from platform to platform, of course. The same was, and still is, true for other functionality that relies on interacting with the environment provided by the platform that the program runs on, e.g. file I/O. Compilers provided such functionality in ways that were unspecified. The only specification was of the API that a programmer used to access these features.
- ³ Today, 2D computer graphics displays have replaced console I/O as the dominant interactive I/O technology. For example, writing a simple `cout << "Hello, world!";` statement doesn't do anything useful on most tablets and smartphones. Even on PCs, console I/O is generally a legacy technology rather than the primary form of user interaction.
- ⁴ Thus the absence of 2D computer graphics I/O in the standard library leaves standard C++ less useful and relevant than it was when it was created. In other areas, standard C++ has modernized, evolved, and continues to develop and improve as new features are added. As a result, standard C++ is a vibrant and highly relevant programming language. Except when it comes to providing programmers access to the current dominant interactive I/O technology.
- ⁵ Standard C++ has lost the capability of providing access to the dominant interactive I/O technology, which it had provided when the language was created. The purpose of this Technical Specification is to rectify this. This is a complex subject matter. The design has undergone numerous reviews and revisions since it the initial proposal, N3888, in January 2014. Work on output is now substantially complete. Since the design of an input API is inherently reliant on the design of the output API, from the outset the authors decided to only pursue of the input API after the output API design was complete. This is why there is no proposed input API at this time.

0.1.2 Brief comparison of I/O technologies

[io2d.technologies]

- ¹ For purposes of clarity, we will briefly touch on the differences between console I/O and 2D computer graphics I/O.
- ² Console I/O allows for the input and output of strings, whether as single characters or as multi-line constructs.
- ³ For output purposes, a console is defined in terms of the number of characters it can display per horizontal line, e.g. 80, and the number of horizontal lines of characters, e.g. 24, that it can display. Its finest granularity is at the character level; in other words a programmer can, at most, control the output at the character level. The actual size of those characters on the console in terms of specific pixels is an implementation detail that is inaccessible to the programmer. Individual pixels are not addressable.
- ⁴ For output purposes, a 2D computer graphics display is defined in terms of the number of pixels it can display per horizontal line, e.g. 1920, and the number of horizontal lines of pixels, e.g. 1080, that it can display, called its resolution. It is also defined by the number of bits per pixel, representing the amount of color information each pixel is capable of carrying and thus the amount of variation in color that can be displayed to the user.
- ⁵ The display of characters on 2D computer graphics displays is usually accomplished using technologies that process those characters, transform them into pixels in a manner specified by fonts, and modify the appropriate pixels.
- ⁶ Fonts describe characters in ideal representations and the process of transforming them into pixels is called rendering. Rendering converts the ideal representation into a representation composed of pixels. They are then rasterized, i.e. transformed from their rendered composition of pixels into pixels on the 2D graphics display such that they are sized appropriately based on the program's specification.

- 7 Characters, by way of fonts, are not the only things that can be rendered. Any specification of an ideal representation of graphical data can be rendered into a composition of pixels and the result can then be rasterized. Because different displays have different resolutions, there doesn't need to be a one-to-one ratio of pixels between the rendered result and the rasterization of it. Non-exact rasterization can be, and often is, done, and the method for doing it is defined by any one of a number of algorithms, each of which produce outputs that depend on the values supplied for parameters that they use.
- 8 From an input perspective, the lowest resolution for console input is a character. While lower resolutions are not inherently prohibited, because the lowest resolution of console output is a character, in practice this is also the lowest resolution of input to a console.
- 9 For a 2D computer graphics display, the lowest resolution is a pixel on the display itself. This does not prohibit higher input resolutions, including characters and strings. It is in no way uncommon for a 2D computer graphics display to receive character or string input. It is simply a fact that such displays are virtually always designed to be capable of receiving input at a lower resolution than the character level if that is what the program requires.

0.1.3 Computer graphics history

[io2d.compgraphicshistory]

- 1 Computer graphics first appeared in the 1950s. The first displays were oscilloscopes which could be used to plot points and lines. Spacewar!, which was first released in 1962 for the DEC PDP-1, is widely recognized as the first computer game that was distributed to and played at multiple computer facilities.
- 2 As computers became less exotic and computer time became less expensive, games and puzzles became a typical way for students to learn programming. The first commercial video game, Pong, was a TTL device which rendered a small number of lines and points to a CRT device.
- 3 In 1974 the Evans and Sutherland frame buffer debuted which allowed the display of 512x512 pixel images. Although enormously expensive at \$15,000 (not adjusted for inflation), prices were driven down over time. This new technology allowed raster display to be incorporated into the nascent home-brew computers of the late 1970s.
- 4 With the invention of VRAM in the mid-1980s frame buffers became significantly cheaper and as a result available pixel resolutions increased and color displays steadily replaced monochrome displays. Computer displays now frequently deliver resolutions exceeding 2 million pixels with 24 bits or more of color information.
- 5 In the late 1980s, software programs started to rely on 2D graphics for intuitive feedback of information using spatial contexts on screens. The widespread introduction of home computers to the market made 2D graphics a familiar experience. Many of today's programmers had their first experience of programming on these machines, learning to code by writing graphics demonstrations and simple games.
- 6 During the 1990s graphics co-processors started to appear; these were add-in cards which provided additional computing power for performing vector calculations. They often contained their own separate RAM used for their own frame buffer to preserve locality in hardware. Over the past twenty years available resolutions have come to greatly exceed the typical resolution of a domestic television set.
- 7 Rendering images has spawned a large field of academic study, advanced by Bresenham's line drawing algorithm in 1965, and also by K. Vesprille's dissertation on the B-Spline approximation form in 1975. As colour depths and available grey scales have increased, font rendering and anti-aliasing have become rich areas of investigation. There are entire conferences devoted to rendering, for example SIGGRAPH.

0.1.4 C++ and computer graphics

[io2d.cppcompgraphics]

- 1 Many C++ 2D graphics libraries have been released in the years since computer graphics have become common. Some are very feature-rich, requiring many hundreds of hours of study and use to master. Some support hardware acceleration using the graphics co-processors that have become ubiquitous in many devices. Some support only one OS or only one type of GPU.
- 2 What these libraries do not provide is a standardized C++ API. The C++ Standard Library and the C++ Technical Specifications all provide standard C++ APIs. Some features, such as atomics, rely on the host environment and quality of implementation to determine their ability to work and their performance. The same will be true for 2D graphics. Unlike many other features being added to C++, 2D graphics has a core set of functionality that has existed and has been in wide use for decades.
- 3 Beginning with the PostScript language and continuing through to modern C and C++ libraries such as Skia, is the functionality of plotting points, drawing lines and curves, displaying bitmaps, and rendering text. This functionality has been the stable core of 2D graphics for nearly 40 years (since the advent of PostScript,

if not before then). While the best methods of implementing it has changed over the years as computer hardware has evolved, the functionality itself has remained the stable core of 2D graphics; any other 2D graphics operations can be performed using it. As such there is no reason to be concerned about the future utility of a standard C++ API that embodies this functionality. 2D graphics is, in effect, a solved problem. Indeed, even the rendering of text is performed either by displaying bitmaps (using bitmap fonts) or by rendering points, lines, and curves (using text rendering descriptions such as OpenType fonts), such that text rendering, while sometimes considered a core part of 2D graphics functionality, is in fact a superset of the true core of 2D graphics.

0.1.5 Goals for the proposed TS

[io2d.goals]

- 1 Get feedback from implementers regarding changes that could be made to allow them to provide better performance. This design decouples the front end user API and the back end implementation by using templates, thus mimicking a standard ABI as close as possible. Back end implementations will still need to provide builds that are compatible with each linker that they want to allow users to make use of. But the front end is able to be provided as non-compiled header files. In fact the reference implementation already provides a front end implementation that is licensed under terms that allow all its use by commercial and non-commercial implementations, such that there is no need for implementers to implement the front end unless they are restricted from doing so by policies they are required to follow.
- 2 Get feedback from users regarding improvements in usability and requests for additional features. While the design has undergone numerous design reviews and revisions, feedback from the C++ community is crucial to ensuring that this API meets the needs of the community at large. As with any library functionality, there will be users who have needs that cannot be met except by an extremely targeted design meant for one or more specific environments. The goal here is to identify areas where a design change would broaden the C++ programming community's ability to use this API in real world products and to make it more usable for C++ programmers who would already be able to use it to meet their needs.

0.2 Revision history

[io2d.revisionhistory]

0.2.1 Revision 10

[io2d.revisionhistory.r10]

- 1 Added responses to P1062R0 and P1225R0, as requested during review in SG13 at the Cologne 2019 meeting. The responses can be found as informative appendices to the proposal. It is anticipated that they will be removed in a future revision.
- 2 Added the `GraphicsSurfaces::text` requirements that were unintentionally absent from Revision 9.
- 3 Clarified how text rendering handles a situation where one or more characters does not map to a glyph in the font being used to render the text. As a result, the purpose of a `basic_font_object`'s *merging* value is now defined.
- 4 The `basic_font` constructor that has a `std::filesystem::path` parameter now also has a `string` family parameter. This addition was made because OFF Font Format Font Collections may contain multiple families and in such cases users should specify the family they desire rather than having the implementation always automagically select one for them.
- 5 Renamed the `basic_matrix_2d` static factory functions from `init_*` to `create_*`. This change was made in the reference implementation in May 2018 to create naming uniformity but inadvertently was not changed in this document.
- 6 Added missing requirements for `GraphicsMath` matrix `mXY` functions.
- 7 Cleaned up specifications of `basic_brush_props`, `basic_fill_props`, `basic_mask_props`, and `basic_stroke_props`.
- 8 Added member data of type `antialias` to `basic_stroke_props`.
- 9 Removed member data of type `fill_rule` from `basic_brush_props`.
- 10 Updated the descriptions of the stroking and filling operations to reflect the change of the `fill_rule` location and the application of the antialiasing algorithm of type `antialias` to those operations described in the relevant property.
- 11 Changed stable name `[io2d.imagesurface.mofifiers]` to `[io2d.imagesurface.modifiers]`. Changed stable name `[io2d.outputsurface.mofifiers]` to `[io2d.outputsurface.modifiers]`. Changed stable name `[io2d.unmanagedoutputsurface.mofifiers]` to `[io2d.unmanagedoutputsurface.modifiers]`.

- 12 Updated the `fill` member functions and static requirements functions to have a `const optional<basic_fill_props<GraphicsSurfaces>>& fp` parameter seeing as the `basic_fill_props` type was designed to be used by those functions.
- 13 Miscellaneous typo and formatting fixes.

0.2.2 Revision 9

[io2d.revisionhistory.r9]

- 1 Text rendering has been added to the surface classes.
- 2 A command list interface for drawing has been added. This allows advanced users to batch drawing commands and submit them to a surface. It can be used in addition to or in lieu of the existing draw callback mechanism.
- 3 The SVG 1.1 Standard format is now included as an `image_file_format` enumerator.
- 4 This Clause is now an Introduction clause as defined by ISO/IEC directives. It now includes informative information and commentary about the technical content of this Technical Specification and the reasons prompting its preparation. Much of it is drawn from information previously published in P0669R0, with revisions and expansions. This is a result of feedback from the San Diego 2018 conference where it was noted that it is advisable to include this sort of information directly in this Technical Specification rather than in separate papers.
- 5 There is now an informative clause that describes the design of the library so that implementers can more easily understand how the pieces of this Technical Specification work and how they interact with each other.
- 6 Formatting and typo fixes.
- 7 `rgba_color` is no longer premultiplied. Users can apply premultiplication themselves if they wish. This change allowed the setters to directly set a value rather than be modified by the alpha value, or in the case of the alpha setter modifying all other values.
- 8 The solid color `basic_brush` ctor now takes its `rgba_color` argument by value rather than reference.

0.2.3 Revision 8

[io2d.revisionhistory.r8]

- 1 Modified the revision 7 notes (0.2.4) to denote trademarks where applicable, and to use the correct capitalization for the cairo graphics library. The contents of those notes is otherwise unchanged.
- 2 Changed the Revision history Clause to be Clause 0.
- 3 Added a new Clause, Graphics math (Clause 8), which defines the requirements of a type that conforms to the GraphicsMath template parameter used by various classes.
- 4 Updated the relevant class member functions in this proposal to define their effects to include calls to the appropriate GraphicsMath functions. This completes the work, begun in P0267R7, of abstracting the implementation of the linear algebra and geometry classes, thereby allowing users to specify a preferred implementation of the mathematical functionality used in this proposal.
- 5 Added a new Clause, Graphics surfaces (Clause 9), which defines the requirements of a type that conforms to the GraphicsSurfaces template parameter used by various classes.
- 6 Updated the relevant class member functions in this proposal to define their effects to include calls to the appropriate GraphicsSurfaces functions. This completes the work, begun in P0267R7, of abstracting the implementation of the brush, paths, surface state, and surface classes, thereby allowing users to specify a preferred implementation of the functionality specified in this proposal.
- 7 Added a new Clause, Surface state props (Clause 15) and moved the relevant `enum class` types and the `basic_render_props`, `basic_brush_props`, `basic_clip_props`, `basic_stroke_props`, and `basic_mask_props` class templates to it.
- 8 Added Michael Kazakov as a co-author. He has written an implementation of this proposal using the Core Graphics framework of Cocoa®, thus providing a native implementation for iOS® and OS X®. It is available as part of the reference implementation (See 0.2.4).
- 9 He has also written a series of tests for compliance. This has drawn attention to several issues that have require some revision.
- 10 Eliminated `format::rgb16_565` and `format::rgb30`.
- 11 Eliminated `compositing_op::dest` since it is a no-op.
- 12 Significant cleanup of terms and definitions.

- 13 Added overload of `copy_surface` for `basic_output_surface`.
- 14 Removed `format_stride_for_width`; it has had no use since mapping functionality was removed.
- 15 Added functions `degrees_to_radians` and `radians_to_degrees`.
- 16 Added equality comparison operators for a number of classes.
- 17 Removed the copyright notice that stated that the proposal was copyrighted by ISO/IEC. Neither organization, jointly or severally, made any contribution to this document and no assignment of interests by the authors to either organization, jointly or severally, has ever been executed. The notice was there unintentionally and its presence in all revisions of P0267 was a mistake.
- 18 Added `basic_dashes` which was added in R7 but had its description omitted accidentally.
- 19 Removed the mandate of underlying layout of pixel formats in `enum class format` and made it and, the interpretation of the data (i.e. what each bit value in each channel means), and whether data is in a premultiplied format implementation defined.
- 20 Added `GraphicsSurfaces::additional_formats` This allows implementations to support additional visual data formats.
- 21 Eliminated all `flush` and `mark_dirty` member functions. These only existed to allow users to modify surfaces externally. Implementations that wish to allow users to modify surfaces externally should provide and document their own functionality for how to do that. The errors, etc., are all implementation dependent anyway so a uniform calling interface provides no benefit at all in the current templated-design.
- 22 Renamed `enum class refresh_rate` to `refresh_style` to more accurately reflect its meaning. This was already done in parts of the R7; it is now complete.
- 23 Changed the order of items in the `basic_figure_items::figure_item` type alias from alphabetical to grouping by function (e.g. `abs_new_figure`, `rel_new_figure`, and `close_figure` are grouped together and `abs_line` and `rel_line` are grouped together). While it's not expected that any new figure item types will be added, there is no chance that the existing ones will be augmented with additional types. So if new figure items are added, grouping by type will simply add them to the end, thus preserving the validity of the existing index values without having the existing entries be alphabetized and new entries not being alphabetized.
- 24 Moved the class template definitions for the nested classes within `basic_figure_items<GraphicsSurfaces>` to the descriptions of each of those types from the synopsis of `basic_figure_items<GraphicsSurfaces>` itself.
- 25 Added `format::xrgb16`. The number of bits per channel is left to the implementation since, e.g., Windows® is 565 whereas OS X® and iOS® are 555 with an unused bit. This is useful for platforms with limited memory where supported so having it as an official enumerator will help.
- 26 Users can now request a different output device format when calling the overloads of the `basic_output_surface` ctor and the `basic_unmanaged_output_surface` ctor that take separate output device width and height preferences.
- 27 Eliminated `redraw_required` from `basic_unmanaged_output_surface`. Users can and should track the need to redraw in their own code when they manage the output device.
- 28 Eliminated `user_scaling_callback` functionality from `basic_output_surface` and `basic_unmanaged_output_surface` since the output device is intentionally not fully specified (same as `stdout`, etc.).
- 29 `begin_show` now returns `void` instead of `int` and has an `error_code` overload in case the user tries to show more output surfaces than the system permits.
- 30 `render_props` now has a `filter` instead of an `antialias`.
- 31 `stroke_props` now has an `antialias` instead of a `filter`.
- 32 New type `basic_fill_props` for parameters specific to the fill operation.
- 33 Removed the `fill_rule` from `basic_brush_props` as it was only being used for fill operations.

0.2.4 Revision 7

[io2d.revisionhistory.r7]

- 1 The significant difference between R7 and R6 is the abstraction of the implementation into separate classes. These classes provide math and rendering support. The linear algebra and geometry classes are templated over any appropriate math support class, while the path, brush and surface classes are templated over any appropriate rendering support class.

- ² The reference implementation of this paper provides a software implementation of the math and rendering support classes. This is based on cairo; indeed, so far the reference implementation has been based on cairo. However, it is now possible to provide an implementation more appropriate to the target platform.
- ³ For example, a Windows® implementation could provide support classes based on DirectX®, while a Linux® implementation could provide support classes based on OpenGL®. In fact, any hardware vendor could provide a support library, targeting a specific implementation and their particular silicon if they wanted to exploit particular features of their hardware.
- ⁴ Additionally, the surface classes have been modified: now there are simply managed and unmanaged output surfaces, the latter of which offers developers the opportunity to take finer control of the drawing surface
- ⁵ The modified classes are as follows

Table 1 — Class identifiers modified since R6

R6 identifier	R7 identifier
vector_2d	basic_point_2d
matrix_2d	basic_matrix_2d
rectangle	basic_bounding_box
circle	basic_circle
path_group	basic_interpreted_path
path_builder	basic_path_builder
color_stop	gradient_stop
brush	basic_brush
render_props	basic_render_props
brush_props	basic_brush_props
clip_props	basic_clip_props
stroke_props	basic_stroke_props
mask_props	basic_mask_props
image_surface	basic_image_surface
display_surface	basic_output_surface

- ⁶ The `surface` class and the `mapped_surface` class have been withdrawn, while the `basic_unmanaged_output_surface` class has been introduced.
- ⁷ The reference implementation, including a software-only implementation of math and rendering support classes, is available at https://github.com/mikebmcl/P0267_RefImpl

0.2.5 Revision 6

[io2d.revisionhistory.r6]

- ¹ Presented to LEWG in Toronto, July 2017

1 Scope

[io2d.scope]

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

2 Normative references

[io2d.refs]

¹ The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- (1.1) — ISO/IEC 14882, *Programming languages — C++*
- (1.2) — ISO/IEC 2382 (all parts), *Information technology — Vocabulary*
- (1.3) — ISO/IEC 10646, *Information technology — Universal Coded Character Set (UCS)*
- (1.4) — ISO/IEC 10918-1, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines*
- (1.5) — ISO 12639, *Graphic technology – Prepress digital data exchange – Tag image file format for image technology (TIFF/IT)*
- (1.6) — ISO/IEC 14496-22 *Information technology – Coding of audio-visual objects – Open Font Format*
- (1.7) — ISO/IEC 15948 *Information technology – Computer graphics and image processing – Portable Network Graphics (PNG) Functional specification*
- (1.8) — ISO/IEC TR 19769:2004, *Information technology — Programming languages, their environments and system software interfaces — Extensions for the programming language C to support new character data types*
- (1.9) — ISO 15076-1, *Image technology colour management — Architecture, profile format and data structure — Part 1: Based on ICC.1:2004-10*
- (1.10) — IEC 61966-2-1, *Colour Measurement and Management in Multimedia Systems and Equipment - Part 2-1: Default RGB Colour Space - sRGB*
- (1.11) — ISO 32000-1:2008, *Document management — Portable document format — Part 1: PDF 1.7*
- (1.12) — ISO 80000-2:2009, *Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology*
- (1.13) — Tantek Çelik et al., *CSS Color Module Level 3 — W3C Recommendation 19 June 2018*, Copyright © 2019 W3C® (MIT, ERCIM, Keio, Beihang) [Viewed 2019-06-12]. Available at <https://www.w3.org/TR/2018/REC-css-color-3-20180619/>.
- (1.14) — Daggett, John et al., *CSS Fonts Module Level 3 — W3C Recommendation 20 September 2018*, Copyright © 2018 W3C® (MIT, ERCIM, Keio, Beihang) [Viewed 2019-06-12]. Available at <https://www.w3.org/TR/2018/REC-css-fonts-3-20180920/>.
- (1.15) — Dahlström, Erik et al., *Scalable Vector Graphics (SVG) 1.1 (Second Edition) — W3C Recommendation 16 August 2011*, Copyright © 2011 W3C® (MIT, ERCIM, Keio) [Viewed 2019-06-12]. Available at <http://www.w3.org/TR/2011/REC-SVG11-20110816/>.

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

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

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

⁶ The document CSS Color Module Level 3 — W3C Recommendation 19 June 2018 is hereinafter called the *CSS Colors Specification*.

⁷ The UTF-8 encoding scheme described in ISO/IEC 10646 is hereinafter called *UTF-8*.

⁸ The open font format described in ISO/IEC 14496-22 is hereinafter called the *OFF Font Format*.

⁹ The document CSS Fonts Module Level 3 — W3C Recommendation 20 September 2018 is hereinafter called the *CSS Fonts Specification*.

¹⁰ The document Scalable Vector Graphics (SVG) 1.1 (Second Edition) — W3C Recommendation 16 August 2011 is hereinafter called the *SVG 1.1 Standard*.

3 Terms and definitions [io2d.defns]

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

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

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

3.1 [io2d.defns.point]

point

coordinate designated by a floating-point x axis value and a floating-point y axis value

3.2 [io2d.defns.origin]

origin

point with an x axis value of 0 and a y axis value of 0

3.3 [io2d.defns.stndcrdspace]

standard coordinate space

Euclidean plane described by a Cartesian coordinate system where the x axis is a horizontal axis oriented from left to right, the y axis is a vertical axis oriented from top to bottom, and rotation of a point, excluding the origin, around the origin by a positive value in radians is counterclockwise

3.4 [io2d.defns.point.integral]

integral point

point where the x axis value and the y axis value are integers

3.5 [io2d.defns.normalize]

normalize

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 $[-1, 1]$ is not used in this Technical Specification . — *end note*]

3.6 [io2d.defns.aspectratio]

aspect ratio

ratio of the width to the height of a rectangular area

3.7 [io2d.defns.visdata]

visual data

data in a possibly bounded Euclidean plane consisting of one or more components representing color, transparency, or some other quality where the component values are not necessarily uniform throughout the plane

3.8 [io2d.defns.visdataelem]

visual data element

unit of visual data at a specific point

3.9 [io2d.defns.channel]

channel

component of visual data

3.10 [io2d.defns.colorchannel]

color channel

channel that only represents the intensity of a specific color

- 3.11** [io2d.defns.alphachannel]
alpha channel
channel that only represents transparency
- 3.12** [io2d.defns.visdatafmt]
visual data format
specification of information necessary to transform a set of one or more channels into colors in a color model
- 3.13** [io2d.defns.premultipliedformat]
premultiplied format
visual data format with one or more color channels and an alpha channel where each color channel is normalized and then multiplied by the normalized alpha channel value [*Example:* Given the 32-bit non-premultiplied RGBA pixel with 8 bits per channel {255, 0, 0, 127} (half-transparent red), when normalized it would become {1.0f, 0.0f, 0.0f, 0.5f}. When premultiplied it would become {0.5f, 0.0f, 0.0f, 0.5f} as a result of multiplying each of the three color channels by the alpha channel value. — *end example*]
- 3.14** [io2d.defns.rastergfxdata]
raster graphics data
data comprised of a rectangular array of visual data elements together with their visual data format where the top-left visual data element is located at the origin in the standard coordinate space and additional visual data elements are located at integral points of consecutive values
- 3.15** [io2d.defns.pixel]
pixel
discrete element of raster graphics data
- 3.16** [io2d.defns.vectorgfxdata]
vector graphics data
data comprised of zero or more paths together with a sequence of rendering and composing operations and graphics state data that produces continuous visual data when processed
- 3.17** [io2d.defns.colormodel]
color model
ideal, mathematical representation of color
- 3.18** [io2d.defns.additivecolor]
additive color
color defined by the emissive intensity of its color channels
- 3.19** [io2d.defns.rgbcolormodel]
RGB color model
color model using additive color comprised of red, green, and blue color channels
- 3.20** [io2d.defns.rgbacolormodel]
RGBA color model
RGB color model with an alpha channel
- 3.21** [io2d.defns.colorsapce]
color space
systematic mapping of values to colorimetric colors
- 3.22** [io2d.defns.srgbcolourspace]
sRGB color space
color space defined in IEC 61966-2-1 that is based on the RGB color model
- 3.23** [io2d.defns.startpt]
start point
point that begins a segment

- 3.24** [io2d.defns.endpt]
end point
 point that ends a segment
- 3.25** [io2d.defns.controlpt]
control point
 point, other than the start point and the end point, that is used in defining a curve
- 3.26** [io2d.defns.bezier.quadratic]
Bézier curve
 ⟨quadratic⟩ curve defined by the equation $f(t) = (1-t)^2 \times P_0 + 2 \times t \times (1-t) \times P_1 + t^2 \times t \times P_2$ where t is in the range $[0, 1]$, P_0 is the start point, P_1 is the control point, and P_2 is end point
- 3.27** [io2d.defns.bezier.cubic]
Bézier curve
 ⟨cubic⟩ curve defined by the equation $f(t) = (1-t)^3 \times P_0 + 3 \times t \times (1-t)^2 \times P_1 + 3 \times t^2 \times (1-t) \times P_2 + t^3 \times t \times P_3$ where t is in the range $[0, 1]$, P_0 is the start point, P_1 is the first control point, P_2 is the second control point, and P_3 is the end point
- 3.28** [io2d.defns.seg]
segment
 line, Bézier curve, or arc
- 3.29** [io2d.defns.initialseg]
initial segment
 segment in a figure whose start point is not defined as being the end point of another segment in the figure [*Note*: It is possible for the initial segment and final segment to be the same segment. — *end note*]
- 3.30** [io2d.defns.newfigpt]
new figure point
 point that is the start point of the initial segment
- 3.31** [io2d.defns.finalseg]
final segment
 segment in a figure whose end point does not define the start point of any other segment [*Note*: It is possible for the initial segment and final segment to be the same segment. — *end note*]
- 3.32** [io2d.defns.currentpt]
current point
 point used as the start point of a segment
- 3.33** [io2d.defns.openfigure]
open figure
 figure with one or more segments where the new figure point is not used to define the end point of the figure's final segment [*Note*: Even if the start point of the initial segment and the end point of the final segment are assigned the same coordinates, the figure is still an open figure. This is because the final segment's end point is not defined as being the new figure point but instead merely happens to have the same value as that point. — *end note*]
- 3.34** [io2d.defns.closedfigure]
closed figure
 figure with one or more segments where the new figure point is used to define the end point of the figure's final segment
- 3.35** [io2d.defns.degenerateseg]
degenerate segment
 segment that has the same values for its start point, end point, and, if any, control points

- 3.36** [io2d.defns.command.closefig]
command
 ⟨close figure command⟩ instruction that creates a line segment with a start point of current point and an end point of new figure point
- 3.37** [io2d.defns.command.newfig]
command
 ⟨new figure command⟩ an instruction that creates a new path
- 3.38** [io2d.defns.figitem]
figure item
 segment, new figure command, close figure command, or path command
- 3.39** [io2d.defns.figure]
figure
 collection of figure items where the end point of each segment in the collection, except the final segment, defines the start point of exactly one other segment in the collection
- 3.40** [io2d.defns.path]
path
 collection of figures
- 3.41** [io2d.defns.pathtransform]
path transformation matrix
 affine transformation matrix used to apply affine transformations to the points in a path
- 3.42** [io2d.defns.pathcommand]
path command
 instruction that modifies the path transformation matrix
- 3.43** [io2d.defns.degeneratefigure]
degenerate figure
 figure containing a new figure command, zero or more degenerate segments, zero or more path commands, and, optionally, a close figure command
- 3.44** [io2d.defns.graphicsstatedata]
graphics state data
 data which specify how some part of the process of rendering or composing is performed in part or in whole
- 3.45** [io2d.defns.render]
render
 transform a path into visual data
- 3.46** [io2d.defns.compositionalgorithm]
composition algorithm
 algorithm that combines source visual data and destination visual data producing visual data that has the same visual data format as the destination visual data
- 3.47** [io2d.defns.compose]
compose
 apply a composition algorithm
- 3.48** [io2d.defns.renderingandcomposingop]
rendering and composing operation
 operation that is either a composing operation or a rendering operation followed by a composing operation that uses the data produced by the rendering operation
- 3.49** [io2d.defns.filter]
filter
 algorithm that determines a color value from a raster graphics data source for a non-integral point

3.50**sample**

apply a filter

[io2d.defns.sample]

3.51**aliasing**

errors in the appearance of the results of rendering where the resulting visual data is raster graphics data because of inaccuracies in transforming continuous data into discrete data

[io2d.defns.alias]

3.52**anti-aliasing**

application of an algorithm while rendering to reduce aliasing

[io2d.defns.antialias]

4 Error reporting

[io2d.err.report]

- ¹ 2D graphics library functions that can produce errors occasionally provide two overloads: one that throws an exception to report errors and another that reports errors using an `error_code` object. This provides for situations where errors are not truly exceptional.
- ² report errors as follows, unless otherwise specified:
- ³ When an error prevents the function from meeting its specifications:
 - (3.1) — Functions that do not take argument of type `error_code&` throw an exception of type `system_error` or of an implementation-defined type that derives from `system_error`. The exception object shall include the enumerator specified by the function as part of its observable state.
 - (3.2) — Functions that take an argument of type `error_code&` assigns the specified enumerator to the provided `error_code` object and then returns.
- ⁴ Failure to allocate storage is reported by throwing an exception as described in [res.on.exception.handling] in C++ 2017.
- ⁵ Destructor operations defined in this Technical Specification shall not throw exceptions. Every destructor in this Technical Specification shall behave as-if it had a non-throwing exception specification.
- ⁶ If no error occurs in a function that takes an argument of type `error_code&`, `error_code::clear` shall be called on the `error_code` object immediately before the function returns.
- ⁷ Where the specification of a function template, including member functions of class templates, declares that there may be implementation-defined errors, the implementer is the provider of the type used as the template argument, referred to at times as the "back end" (see: [Clause 5](#), [Clause 8](#), and [Clause 9](#)). The implementer is not the provider of the class template or function template.

5 Library design overview (Informative)

[io2d.design]

- ¹ In order to provide an effective, efficient, standardized 2D graphics library, the library design makes use of templates. The main class templates described provide a standard API "front end" and the template arguments to them provide the implementation "back end".
- ² The front end is described in terms of calling specific functions that back ends are required to provide in order to meet the requirements necessary to be a valid back end. As a result, the front end code can be and likely will be identical for all standard library implementations. As a time saving measure, standard library implementers can likely simply incorporate the source code for the front end that is provided by the reference implementation. The license of the reference implementation is intended to allow this and its authors are happy to work with standard library implementers to try to ensure that this can be done (e.g. by providing the option to license the front end code under a different license if necessary). Regardless, the specification of the front end in this Technical Specification is sufficient for standard library implementers to write their own implementation without any need to refer to the reference implementation.
- ³ The back end, which consists of classes meeting the requirements specified herein, is where the platform-specific operations that make this library work are performed.
- ⁴ Back ends can be provided by standard library implementations and by others. It is expected that standard library implementations will provide back ends for all relevant platforms they support, but they aren't required to do so. This significantly reduces the burden on standard library implementations. Others, in turn, need only provide the back end, not a full standard library implementation.
- ⁵ Users of the library can choose which back end they are using by changing the template arguments provided to the front end and making any other changes required by the compiler and build system they are using.
- ⁶ Only back ends provided by standard library implementers are allowed to be in the `std` namespace. This requirement exists simply to prevent naming collisions that otherwise could occur.
- ⁷ The back end specifies requirements for two classes, one for mathematics functionality required for 2D graphics, `GraphicsMath`, and the other for 2D graphics operations, `GraphicsSurfaces`. The design is such that while `GraphicsSurfaces` uses functionality provided by `GraphicsMath`, the two are independent of each other. This allows users to choose the best ones for their needs.
- ⁸ It is expected that the template arguments provided to the front end will be user defined type aliases. If so, to change back ends, users would change the type aliases, include any back end-specific header files for the desired back end, and use implementation dependent mechanisms, such as arguments to the compiler, to ensure that the back end is available when the program runs.
- ⁹ With conditional type aliases, conditional includes, and conditional arguments to the compiler, the user can easily change back ends without ever needing to change the code that performs the graphics operations, since all of that code exclusively uses the front end. The back end is not meant to be directly invoked by the users, and the description of the library functionality does not provide users with the ability to access the back end via the front end APIs.
- ¹⁰ Two goals are met by adopting this design.
- ¹¹ First, standard library implementers do not need to provide full back ends for every single platform on which their standard library implementation might be used. Standard library implementers are required to provide a back end that, at a minimum, supports all operations on image surfaces. The reason for this is that it is possible to write platform independent code that supports image surfaces and all operations on them (subject to host environment limitations, of course).
- ¹² The reference implementation relies on outside libraries for back end image surface operations such that standard library implementers might not be able to use the reference implementation to meet this requirement. They will need to evaluate how best to meet this requirement, which could require a non-trivial, albeit one-time, expenditure of time to write the necessary code to meet this obligation (not including future maintenance and possible modifications as a result of changes made as a byproduct of the TS process). This is considered a reasonable trade off since it ensures that there is always support for image surfaces (subject

to host limitations) without requiring standard library implementers to take on any new platform-specific dependencies.

- ¹³ Second, users are not forced to use the back ends that standard library implementations do provide. Users can choose their own back ends (whether written in-house or by third parties) for purposes of gaining access to back ends that support other platforms and back ends that provide more performant implementations for specific platforms than what might be provided by standard library implementations.
- ¹⁴ Third parties such as graphics acceleration hardware vendors, operating system vendors, graphics environment vendors, and others can write back ends and make them available to users directly or perhaps even by contributing them to standard library implementations. As hardware and software evolves, new and updated back ends can be provided. Since a back end only needs to conform to the specified requirements, these individuals and organizations are not committed to implementing any part of the C++ standard library. Nor are they required to produce updates quickly (or even at all) should the API change due to the use of inline namespaces to provide versioning for large APIs such as this.
- ¹⁵ In summation, the API of the 2D graphics library is split into two parts: a front end composed of classes, class templates, and scoped enums that is expected to be fairly static in its design and will be what users use for standardized 2D graphics programming in C++; and a back end that is a specification of requirements for classes that are used by the front end to perform the graphics operations specified by the front end. Standard library implementers are required to provide the front end and a back end that provides only the platform-independent functionality required by the requirements for a back end. Back ends are provided as template arguments to the class templates of the front end such that users can choose back ends provided by standard library implementations, by third parties, or by the user's own back end implementation that meet the user's needs.

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

```

namespace std::experimental::io2d {
    inline namespace v1 {
        template <class T>
        constexpr T pi = T(3.14159265358979323846264338327950288L);
        template <class T>
        constexpr T two_pi = T(6.28318530717958647692528676655900577L);
        template <class T>
        constexpr T half_pi = T(1.57079632679489661923132169163975144L);
        template <class T>
        constexpr T three_pi_over_two = T(4.71238898038468985769396507491925432L);
        template <class T>
        constexpr T tau = T(6.28318530717958647692528676655900577L);
        template <class T>
        constexpr T three_quarters_tau = T(4.71238898038468985769396507491925432L);
        template <class T>
        constexpr T half_tau = T(3.14159265358979323846264338327950288L);
        template <class T>
        constexpr T quarter_tau = T(1.57079632679489661923132169163975144L);

        template <class T>
        constexpr T degrees_to_radians(T deg) noexcept;
        template <class T>
        constexpr T radians_to_degrees(T rad) noexcept;

        class rgba_color;
        constexpr bool operator==(const rgba_color& lhs, const rgba_color& rhs)
            noexcept;
        constexpr bool operator!=(const rgba_color& lhs, const rgba_color& rhs)
            noexcept;
        template <class T>
        constexpr rgba_color operator*(const rgba_color& lhs, T rhs) noexcept;
        template <class U>
        constexpr rgba_color operator*(const rgba_color& lhs, U rhs) noexcept;
        template <class T>
        constexpr rgba_color operator*(T lhs, const rgba_color& rhs) noexcept;
        template <class U>
        constexpr rgba_color operator*(U lhs, const rgba_color& rhs) noexcept;

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

        template <class GraphicsMath>
        class basic_bounding_box;
        template <class GraphicsMath>
        bool operator==(const basic_bounding_box<GraphicsMath>& lhs,
            const basic_bounding_box<GraphicsMath>& rhs) noexcept;
        template <class GraphicsMath>
        bool operator!=(const basic_bounding_box<GraphicsMath>& lhs,
            const basic_bounding_box<GraphicsMath>& rhs) noexcept;
    }
}

```

```

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

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

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

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

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

template <class GraphicsMath>
class basic_display_point;

template <class GraphicsSurfaces>
class basic_figure_items;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& rhs) noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& rhs)
    noexcept;

```



```

template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& rhs)

```

```

    noexcept;
template <class GraphicsSurfaces>
bool operator==(const typename
    basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator!=(const typename
    basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator==(const typename
    basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator!=(const typename
    basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator==(const typename
    basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator!=(const typename
    basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator==(const typename
    basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator!=(const typename
    basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve&
    rhs) noexcept;
template <class GraphicsSurfaces>
bool operator==(const typename basic_figure_items<GraphicsSurfaces>::arc&
    lhs, const typename basic_figure_items<GraphicsSurfaces>::arc& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(const typename basic_figure_items<GraphicsSurfaces>::arc&
    lhs, const typename basic_figure_items<GraphicsSurfaces>::arc& rhs)
    noexcept;

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

template <class GraphicsSurfaces>
class basic_image_surface;

template <class GraphicsSurfaces>
class basic_interpreted_path;

```

```

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

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

template <class GraphicsMath>
class basic_matrix_2d;
template <class GraphicsMath>
basic_matrix_2d<GraphicsMath> operator*(
  const basic_matrix_2d<GraphicsMath>& lhs,
  const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(
  const basic_point_2d<GraphicsMath>& lhs,
  const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator==(const basic_matrix_2d<GraphicsMath>& lhs,
  const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator!=(const basic_matrix_2d<GraphicsMath>& lhs,
  const basic_matrix_2d<GraphicsMath>& rhs) noexcept;

template <class GraphicsSurfaces>
class basic_output_surface;

template <class GraphicsSurfaces, class Allocator =
  allocator<typename basic_figure_items<GraphicsSurfaces>::figure_item>>
class basic_path_builder;
template <class GraphicsSurfaces, class Allocator>
bool operator==(const basic_path_builder<GraphicsSurfaces, Allocator>& lhs,
  const basic_path_builder<GraphicsSurfaces, Allocator>& rhs) noexcept;
template <class GraphicsSurfaces, class Allocator>
bool operator!=(const basic_path_builder<GraphicsSurfaces, Allocator>& lhs,
  const basic_path_builder<GraphicsSurfaces, Allocator>& rhs) noexcept;
template <class GraphicsSurfaces, class Allocator>
void swap(basic_path_builder<GraphicsSurfaces, Allocator>& lhs,
  basic_path_builder<GraphicsSurfaces, Allocator>& rhs) noexcept(noexcept(lhs.swap(rhs)));

template <class GraphicsMath>
class basic_point_2d;
template <class GraphicsMath>
bool operator==(const basic_point_2d<GraphicsMath>& lhs,
  const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator!=(const basic_point_2d<GraphicsMath>& lhs,
  const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator+(
  const basic_point_2d<GraphicsMath>& val) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator+(
  const basic_point_2d<GraphicsMath>& lhs,

```

```

    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator-(
    const basic_point_2d<GraphicsMath>& val) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator-(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(
    const basic_point_2d<GraphicsMath>& lhs,
    float rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(float lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator/(
    const basic_point_2d<GraphicsMath>& lhs,
    float rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator/(float lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator/(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;

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

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

template <class GraphicsSurfaces>
class basic_unmanaged_output_surface;

using bounding_box = basic_bounding_box<default_graphics_math>;
using brush = basic_brush<default_graphics_surfaces>;
using brush_props = basic_brush_props<default_graphics_surfaces>;
using circle = basic_circle<default_graphics_math>;
using clip_props = basic_clip_props<default_graphics_surfaces>;
using dashes = basic_dashes<default_graphics_surfaces>;
using display_point = basic_display_point<default_graphics_math>;
using figure_items = basic_figure_items<default_graphics_surfaces>;
using fill_props = basic_fill_props<default_graphics_surfaces>;
using image_surface = basic_image_surface<default_graphics_surfaces>;
using interpreted_path = basic_interpreted_path<default_graphics_surfaces>;
using mask_props = basic_mask_props<default_graphics_surfaces>;

```

```

using matrix_2d = basic_matrix_2d<default_graphics_math>;
using output_surface = basic_output_surface<default_graphics_surfaces>;
using path_builder = basic_path_builder<default_graphics_surfaces>;
using point_2d = basic_point_2d<default_graphics_math>;
using render_props = basic_render_props<default_graphics_surfaces>;
using stroke_props = basic_stroke_props<default_graphics_surfaces>;
using unmanaged_output_surface =
    basic_unmanaged_output_surface<default_graphics_surfaces>;

template <class GraphicsSurfaces>
basic_image_surface<GraphicsSurfaces> copy_surface(
    basic_image_surface<GraphicsSurfaces>& sfc) noexcept;
template <class GraphicsSurfaces>
basic_image_surface<GraphicsSurfaces> copy_surface(
    basic_output_surface<GraphicsSurfaces>& sfc) noexcept;
template <class T>
constexpr T degrees_to_radians(T d) noexcept;
template <class T>
constexpr T radians_to_degrees(T r) noexcept;
float angle_for_point(point_2d ctr, point_2d pt) noexcept;
point_2d point_for_angle(float ang, float rad = 1.0f) noexcept;
point_2d point_for_angle(float ang, point_2d rad) noexcept;
point_2d arc_start(point_2d ctr, float sang, point_2d rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
point_2d arc_center(point_2d cpt, float sang, point_2d rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
point_2d arc_end(point_2d cpt, float eang, point_2d rad,
    const matrix_2d& m = matrix_2d{}) noexcept;
} }

```

7 Colors

[io2d.colors]

7.1 Introduction to color

[io2d.colors.intro]

- 1 Color involves many disciplines and has been the subject of many papers, treatises, experiments, studies, and research work in general.
- 2 While color is an important part of computer graphics, it is only necessary to understand a few concepts from the study of color for computer graphics.
- 3 A color model defines color mathematically without regard to how humans actually perceive color. These color models are composed of some combination of channels which each channel representing alpha or an ideal color. Color models are useful for working with color computationally, such as in composing operations, because their channel values are homogeneously spaced.
- 4 A color space, for purposes of computer graphics, is the result of mapping the ideal color channels from a color model, after making any necessary adjustment for alpha, to color channels that are calibrated to align with human perception of colors. Since the perception of color varies from person to person, color spaces use the science of colorimetry to define those perceived colors in order to obtain uniformity to the extent possible. As such, the uniform display of the colors in a color space on different output devices is possible. The values of color channels in a color space are not necessarily homogeneously spaced because of human perception of color.
- 5 Color models are often termed *linear* while color spaces are often termed *gamma corrected*. The mapping of a color model, such as the RGB color model, to a color space, such as the sRGB color space, is often the application of gamma correction.
- 6 Gamma correction is the process of transforming homogeneously spaced visual data to visual data that, when displayed, matches the intent of the untransformed visual data.
- 7 For example a color that is 50% of the maximum intensity of red when encoded as homogeneously spaced visual data, will likely have a different intensity value when it has been gamma corrected so that a human looking at on a computer display will see it as being 50% of the maximum intensity of red that the computer display is capable of producing. Without gamma correction, it would likely have appeared as though it was closer to the maximum intensity than the untransformed data intended it to be.
- 8 In addition to color channels, colors in computer graphics often have an alpha channel. The value of the alpha channel represents transparency of the color channels when they are combined with other visual data using certain composing algorithms. When using alpha, it should be used in a premultiplied format in order to obtain the desired results when applying multiple composing algorithms that utilize alpha.

7.2 Color usage requirements

[io2d.colors.reqs]

- 1 During rendering and composing operations, color data is linear and, when it has an alpha channel associated with it, in premultiplied format. Implementations shall make any necessary conversions to ensure this.

7.3 Class `rgba_color`

[io2d.rgba_color]

7.3.1 `rgba_color` overview

[io2d.rgba_color.intro]

- 1 The class `rgba_color` describes a four channel color using the RGBA color model.
- 2 There are three color channels, red, green, and blue, each of which is a `float` value.
- 3 There is also an alpha channel, which is a `float` value.
- 4 Legal values for each channel are in the range `[0.0f, 1.0f]`.

7.3.2 `rgba_color` synopsis

[io2d.rgba_color.synopsis]

```
namespace std::experimental::io2d::v1 {
    class rgba_color {
    public:
        // 7.3.3, construct/copy/move/destroy:
        constexpr rgba_color() noexcept;
```

```

template <class T>
constexpr rgba_color(T r, T g, T b, T a = static_cast<T>(0xFF)) noexcept;
template <class U>
constexpr rgba_color(U r, U g, U b, U a = static_cast<U>(1.0f)) noexcept;

```

// 7.3.4, modifiers:

```

template <class T>
constexpr void r(T val) noexcept;
template <class U>
constexpr void r(U val) noexcept;
template <class T>
constexpr void g(T val) noexcept;
template <class U>
constexpr void g(U val) noexcept;
template <class T>
constexpr void b(T val) noexcept;
template <class U>
constexpr void b(U val) noexcept;
template <class T>
constexpr void a(T val) noexcept;
template <class U>
constexpr void a(U val) noexcept;

```

// 7.3.5, observers:

```

constexpr float r() const noexcept;
constexpr float g() const noexcept;
constexpr float b() const noexcept;
constexpr float a() const noexcept;

```

// 7.3.6, static members:

```

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

```

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



```

static const rgba_color misty_rose;
static const rgba_color moccasin;
static const rgba_color navajo_white;
static const rgba_color navy;
static const rgba_color old_lace;
static const rgba_color olive;
static const rgba_color olive_drab;
static const rgba_color orange;
static const rgba_color orange_red;
static const rgba_color orchid;
static const rgba_color pale_goldenrod;
static const rgba_color pale_green;
static const rgba_color pale_turquoise;
static const rgba_color pale_violet_red;
static const rgba_color papaya_whip;
static const rgba_color peach_puff;
static const rgba_color peru;
static const rgba_color pink;
static const rgba_color plum;
static const rgba_color powder_blue;
static const rgba_color purple;
static const rgba_color red;
static const rgba_color rosy_brown;
static const rgba_color royal_blue;
static const rgba_color saddle_brown;
static const rgba_color salmon;
static const rgba_color sandy_brown;
static const rgba_color sea_green;
static const rgba_color sea_shell;
static const rgba_color sienna;
static const rgba_color silver;
static const rgba_color sky_blue;
static const rgba_color slate_blue;
static const rgba_color slate_gray;
static const rgba_color slate_grey;
static const rgba_color snow;
static const rgba_color spring_green;
static const rgba_color steel_blue;
static const rgba_color tan;
static const rgba_color teal;
static const rgba_color thistle;
static const rgba_color tomato;
static const rgba_color transparent_black;
static const rgba_color turquoise;
static const rgba_color violet;
static const rgba_color wheat;
static const rgba_color white;
static const rgba_color white_smoke;
static const rgba_color yellow;
static const rgba_color yellow_green;

// 7.3.7, operators
template <class T>
constexpr rgba_color& operator==(T rhs) noexcept;
template <class U>
constexpr rgba_color& operator!=(U rhs) noexcept;
};

// 7.3.7, operators:
constexpr bool operator==(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;
constexpr bool operator!=(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;

```

```

template <class T>
constexpr rgba_color operator*(const rgba_color& lhs, T rhs) noexcept;
template <class U>
constexpr rgba_color operator*(const rgba_color& lhs, U rhs) noexcept;
template <class T>
constexpr rgba_color operator*(T lhs, const rgba_color& rhs) noexcept;
template <class U>
constexpr rgba_color operator*(U lhs, const rgba_color& rhs) noexcept;
}

```

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

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

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

```

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

```

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

3 *Effects:* Constructs an object of type `rgba_color`. The red channel is `r / 255.0f`. The green channel is `g / 255.0f`. The blue channel is `b / 255.0f`. The alpha channel is `a / 255.0f`.

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

```

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

```

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

6 *Effects:* Constructs an object of type `rgba_color`. The red channel is `static_cast<float>(r)`. The green channel is `static_cast<float>(g)`. The blue channel is `static_cast<float>(b)`. The alpha channel is `static_cast<float>(a)`.

7 *Remarks:* This constructor shall not participate in overload resolution unless `is_floating_point_v<U>` is `true`.

7.3.4 rgba_color modifiers [io2d.rgbacolor.modifiers]

```

template <class T>
constexpr void r(T val) noexcept;

```

1 *Requires:* `val >= 0` and `val <= 255`.

2 *Effects:* The red channel is `val / 255.0f`.

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

```

template <class U>
constexpr void r(U val) noexcept;

```

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

5 *Effects:* The red channel is `static_cast<float>(val)`.

6 *Remarks:* This function shall not participate in overload resolution unless `is_floating_point_v<U>` is `true`.

```

template <class T>
constexpr void g(T val) noexcept;

```

7 *Requires:* `val >= 0` and `val <= 255`.

8 *Effects:* The green channel is `val / 255.0f`.

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

```

template <class U>
constexpr void g(U val) noexcept;
10     Requires: val >= 0.0f and val <= 1.0f.
11     Effects: The green channel is static_cast<float>(val).
12     Remarks: This function shall not participate in overload resolution unless is_floating_point_v<U> is
    true.

template <class T>
constexpr void b(T val) noexcept;
13     Requires: val >= 0 and val <= 255.
14     Effects: The blue channel is val / 255.0f.
15     Remarks: This function shall not participate in overload resolution unless is_integral_v<T> is true.

template <class U>
constexpr void b(U val) noexcept;
16     Requires: val >= 0.0f and val <= 1.0f.
17     Effects: The blue channel is static_cast<float>(val).
18     Remarks: This function shall not participate in overload resolution unless is_floating_point_v<U> is
    true.

template <class T>
constexpr void a(T val) noexcept;
19     Requires: val >= 0 and val <= 255.
20     Effects: The alpha channel is val / 255.0f.
21     Remarks: This function shall not participate in overload resolution unless is_integral_v<T> is true.

template <class U>
constexpr void a(U val) noexcept;
22     Requires: val >= 0.0f and val <= 1.0f.
23     Effects: The alpha channel is static_cast<float>(val).
24     Remarks: This function shall not participate in overload resolution unless is_floating_point_v<U> is
    true.

```

7.3.5 rgba_color observers

[io2d.rgbacolor.observers]

```

constexpr float r() const noexcept;
1     Returns: The red channel.

constexpr float g() const noexcept;
2     Returns: The green channel.

constexpr float b() const noexcept;
3     Returns: The blue channel.

constexpr float a() const noexcept;
4     Returns: The alpha channel.

```

7.3.6 rgba_color static members

[io2d.rgbacolor.statics]

1 The alpha value of all of the predefined `rgba_color` static member object in Table 2 is 1.0f except for `transparent_black`, which has an alpha value of 0.0f.

Table 2 — `rgba_color` static members values

Member name	red	green	blue
<code>alice_blue</code>	240	248	255

Table 2 — rgba_color static members values (continued)

Member name	red	green	blue
antique_white	250	235	215
aqua	0	255	255
aquamarine	127	255	212
azure	240	255	255
beige	245	245	220
bisque	255	228	196
black	0	0	0
blanched_almond	255	235	205
blue	0	0	255
blue_violet	138	43	226
brown	165	42	42
burly_wood	222	184	135
cadet_blue	95	158	160
chartreuse	127	255	0
chocolate	210	105	30
coral	255	127	80
cornflower_blue	100	149	237
cornsilk	255	248	220
crimson	220	20	60
cyan	0	255	255
dark_blue	0	0	139
dark_cyan	0	139	139
dark_goldenrod	184	134	11
dark_gray	169	169	169
dark_green	0	100	0
dark_grey	169	169	169
dark_khaki	189	183	107
dark_magenta	139	0	139
dark_olive_green	85	107	47
dark_orange	255	140	0
dark_orchid	153	50	204
dark_red	139	0	0
dark_salmon	233	150	122
dark_sea_green	143	188	142
dark_slate_blue	72	61	139
dark_slate_gray	47	79	79
dark_slate_grey	47	79	79
dark_turquoise	0	206	209
dark_violet	148	0	211
deep_pink	255	20	147
deep_sky_blue	0	191	255
dim_gray	105	105	105
dim_grey	105	105	105
dodger_blue	30	144	255
firebrick	178	34	34
floral_white	255	250	240
forest_green	34	139	34
fuchsia	255	0	255
gainsboro	220	220	220
ghost_white	248	248	248
gold	255	215	0
goldenrod	218	165	32
gray	128	128	128
green	0	128	0
green_yellow	173	255	47

Table 2 — rgba_color static members values (continued)

Member name	red	green	blue
grey	128	128	128
honeydew	240	255	240
hot_pink	255	105	180
indian_red	205	92	92
indigo	75	0	130
ivory	255	255	240
khaki	240	230	140
lavender	230	230	250
lavender_blush	255	240	245
lawn_green	124	252	0
lemon_chiffon	255	250	205
light_blue	173	216	230
light_coral	240	128	128
light_cyan	224	255	255
light_goldenrod_yellow	250	250	210
light_gray	211	211	211
light_green	144	238	144
light_grey	211	211	211
light_pink	255	182	193
light_salmon	255	160	122
light_sea_green	32	178	170
light_sky_blue	135	206	250
light_slate_gray	119	136	153
light_slate_grey	119	136	153
light_steel_blue	176	196	222
light_yellow	255	255	224
lime	0	255	0
lime_green	50	205	50
linen	250	240	230
magenta	255	0	255
maroon	128	0	0
medium_aquamarine	102	205	170
medium_blue	0	0	205
medium_orchid	186	85	211
medium_purple	147	112	219
medium_sea_green	60	179	113
medium_slate_blue	123	104	238
medium_spring_green	0	250	154
medium_turquoise	72	209	204
medium_violet_red	199	21	133
midnight_blue	25	25	112
mint_cream	245	255	250
misty_rose	255	228	225
moccasin	255	228	181
navajo_white	255	222	173
navy	0	0	128
old_lace	253	245	230
olive	128	128	0
olive_drab	107	142	35
orange	255	69	0
orange_red	255	69	0
orchid	218	112	214
pale_goldenrod	238	232	170
pale_green	152	251	152
pale_turquoise	175	238	238

Table 2 — rgba_color static members values (continued)

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

7.3.7 rgba_color operators

[io2d.rgbacolor.ops]

```

template <class T>
constexpr rgba_color& operator*=(T rhs) noexcept;
1     Requires: rhs >= 0 and rhs <= 255.
2     Effects: r(min(r() * rhs / 255.0f, 1.0f)).
3     g(min(g() * rhs / 255.0f, 1.0f)).
4     b(min(b() * rhs / 255.0f, 1.0f)).
5     a(min(a() * rhs / 255.0f, 1.0f)).
     Returns: *this.
6     Remarks: This function shall not participate in overload resolution unless is_integral_v<T> is true.

template <class U>
constexpr rgba_color& operator*=(U rhs) noexcept;
7     Requires: rhs >= 0.0f and rhs <= 1.0f.

```

```

8     Effects: r(min(r() * static_cast<float>(rhs), 1.0f)).
9     g(min(g() * static_cast<float>(rhs), 1.0f)).
10    b(min(b() * static_cast<float>(rhs), 1.0f)).
11    a(min(a() * static_cast<float>(rhs), 1.0f)).
    Returns: *this.
12    Remarks: This function shall not participate in overload resolution unless is_floating_point_v<T> is
    true.

constexpr bool operator==(const rgba_color& lhs, const rgba_color& rhs)
    noexcept;
13    Returns: lhs.r() == rhs.r() && lhs.g() == rhs.g() && lhs.b() == rhs.b() && lhs.a() ==
    rhs.a().

template <class T>
constexpr rgba_color operator*(const rgba_color& lhs, T rhs) noexcept;
14    Requires: rhs >= 0 and rhs <= 255.
15    Returns:
    rgba_color(min(lhs.r() * rhs / 255.0f, 1.0f),
        min(lhs.g() * rhs / 255.0f, 1.0f),
        min(lhs.b() * rhs / 255.0f, 1.0f),
        min(lhs.a() * rhs / 255.0f, 1.0f))
16    Remarks: This function shall not participate in overload resolution unless is_integral_v<T> is true.

template <class U>
constexpr rgba_color& operator*(const rgba_color& lhs, U rhs) noexcept;
17    Requires: rhs >= 0.0f and rhs <= 1.0f.
18    Returns:
    rgba_color(min(lhs.r() * static_cast<float>(rhs), 1.0f),
        min(lhs.g() * static_cast<float>(rhs), 1.0f),
        min(lhs.b() * static_cast<float>(rhs), 1.0f),
        min(lhs.a() * static_cast<float>(rhs), 1.0f))
19    Remarks: This function shall not participate in overload resolution unless is_floating_point_v<U> is
    true.

template <class T>
constexpr rgba_color operator*(T lhs, const rgba_color& rhs) noexcept;
20    Requires: lhs >= 0 and lhs <= 255.
21    Returns:
    rgba_color(min(lhs * rhs.r() / 255.0f, 1.0f),
        min(lhs * rhs.g() / 255.0f, 1.0f),
        min(lhs * rhs.b() / 255.0f, 1.0f),
        min(lhs * rhs.a() / 255.0f, 1.0f))
22    Remarks: This function shall not participate in overload resolution unless is_integral_v<T> is true.

template <class U>
constexpr rgba_color& operator*(U lhs, const rgba_color& rhs) noexcept;
23    Requires: lhs >= 0.0f and lhs <= 1.0f.
24    Returns:
    rgba_color(min(static_cast<float>(lhs) * rhs.r(), 1.0f),
        min(static_cast<float>(lhs) * rhs.g(), 1.0f),
        min(static_cast<float>(lhs) * rhs.b(), 1.0f),
        min(static_cast<float>(lhs) * rhs.a(), 1.0f))
25    Remarks: This function shall not participate in overload resolution unless is_floating_point_v<U> is
    true.

```

8 Graphics math

[io2d.graphmath]

8.1 General

[io2d.graphmath.general]

- ¹ This Clause describes components that are used to describe certain geometric types and to perform certain linear algebra operations. [*Note*: These types are intended for use in 2D graphics input/output operations. They are not meant to provide a full set of linear algebra types and operations. — *end note*]
- ² The following subclauses describe graphics math requirements and components for linear algebra classes and geometry classes, as summarized in Table 3.

Table 3 — Graphics math summary

Subclause	Header(s)
8.2	GraphicsMath traits
Clause 10	Linear algebra classes <experimental/io2d>
Clause 11	Geometry classes <experimental/io2d>

8.2 Requirements

[io2d.graphmath.reqs]

- ¹ This subsection defines requirements on *GraphicsMath* types.
- ² Most classes specified in Clause 10 through Clause 16 need a set of related types and functions to complete the definition of their semantics. These types and functions are provided as a set of member *typedef-names* and **static** member functions in the template parameter **GraphicsMath** used by each such template. This subclause defines the semantics of these members.
- ³ Let **X** be a *GraphicsMath* type.
- ⁴ Table 8 defines required *typedef-names* in **X**, which are identifiers for class types capable of storing all data required to support the corresponding class template.

Table 4 — **X** typedef-names

<i>typedef-name</i>	Class data
<code>bounding_box_data_type</code>	<code>basic_bounding_box<X></code>
<code>circle_data_type</code>	<code>basic_circle<X></code>
<code>display_point_data_type</code>	<code>basic_display_point<X></code>
<code>matrix_2d_data_type</code>	<code>basic_matrix_2d<X></code>
<code>point_2d_data_type</code>	<code>basic_point_2d<X></code>

- ⁵ In order to describe the observable effects of functions contained in Table 6, Table 5 describes the types contained in **X** as-if they possessed certain member data.

Table 5 — GraphicsMath type member data

Type	Member data	Member data type
<code>bounding_box_data_type</code>	<code>x</code>	<code>float</code>
<code>bounding_box_data_type</code>	<code>y</code>	<code>float</code>
<code>bounding_box_data_type</code>	<code>w</code>	<code>float</code>
<code>bounding_box_data_type</code>	<code>h</code>	<code>float</code>
<code>circle_data_type</code>	<code>x</code>	<code>float</code>
<code>circle_data_type</code>	<code>y</code>	<code>float</code>
<code>circle_data_type</code>	<code>r</code>	<code>float</code>
<code>display_point_data_type</code>	<code>x</code>	<code>int</code>
<code>display_point_data_type</code>	<code>y</code>	<code>int</code>

Type	Member data	Member data type
matrix_2d_data_type	m00	float
matrix_2d_data_type	m01	float
matrix_2d_data_type	m02	float
matrix_2d_data_type	m10	float
matrix_2d_data_type	m11	float
matrix_2d_data_type	m12	float
matrix_2d_data_type	m20	float
matrix_2d_data_type	m21	float
matrix_2d_data_type	m22	float
point_2d_data_type	x	float
point_2d_data_type	y	float

- ⁶ In Table 6, B denotes the type `X::bounding_box_data_type`, C denotes the type `X::circle_data_type`, D denotes the type `X::display_point_data_type`, M denotes the type `X::matrix_2d_data_type`, and P denotes the type `X::point_2d_data_type`.
- ⁷ All expressions in Table 6 are `noexcept`. For purposes of brevity, `noexcept` is omitted in the table.

Table 6 — GraphicsMath requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::create_point_2d()</code>	P	Equivalent to: <code>create_point_2d(0.0f, 0.0f);</code>	
<code>X::create_point_2d(float x, float y)</code>	P	Returns an object p.	<i>Postconditions:</i> <code>p.x == x</code> and <code>p.y == y</code> .
<code>X::x(P& p, float x)</code>	void		<i>Postconditions:</i> <code>p.x == x</code> .
<code>X::y(P& p, float y)</code>	void		<i>Postconditions:</i> <code>p.y == y</code> .
<code>X::x(const P& p)</code>	float	Returns <code>p.x</code> .	
<code>X::y(const P& p)</code>	float	Returns <code>p.y</code> .	
<code>X::dot(const P& p1, const P& p2)</code>	float	Returns <code>p1.x * p2.x + p1.y * p2.y</code> .	
<code>X::magnitude(const P& p)</code>	float	Returns <code>sqrt(p.x * p.x + p.y * p.y)</code> .	
<code>X::magnitude_squared(const P& p)</code>	float	Returns <code>p.x * p.x + p.y * p.y</code> .	
<code>X::angular_direction(const P& p)</code>	float	Returns <code>atan2(p.y, p.x) < 0.0f ? atan2(p.y, p.x) + two_pi<float> : atan2(p.y, p.x)</code> .	<i>Remarks:</i> The purpose of adding <code>two_pi<float></code> if the result is negative is to produce values in the range <code>[0.0f, two_pi<float>)</code> .
<code>X::to_unit(const P& p)</code>	P	Returns an object r.	<i>Postconditions:</i> <code>r.x == p.x / magnitude(p)</code> and <code>r.y == p.y / magnitude(p)</code> .

Table 6 — GraphicsMath requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::add(const P& p1, const P& p2)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p1.x + p2.x$ and $r.y == p1.y + p2.y$.
<code>X::add(const P& p, float f)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p.x + f$ and $r.y == p.y + f$.
<code>X::add(float f, const P& p)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == f + p.x$ and $r.y == f + p.y$.
<code>X::subtract(const P& p1, const P& p2)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p1.x - p2.x$ and $r.y == p1.y - p2.y$.
<code>X::subtract(const P& p, float f)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p.x - f$ and $r.y == p.y - f$.
<code>X::subtract(float f, const P& p)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == f - p.x$ and $r.y == f - p.y$.
<code>X::multiply(const P& p1, const P& p2)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p1.x * p2.x$ and $r.y == p1.y * p2.y$.
<code>X::multiply(const P& p, float f)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p.x * f$ and $r.y == p.y * f$.
<code>X::multiply(float f, const P& p)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == f * p.x$ and $r.y == f * p.y$.
<code>X::divide(const P& p1, const P& p2)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p1.x / p2.x$ and $r.y == p1.y / p2.y$.
<code>X::divide(const P& p, float f)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == p.x / f$ and $r.y == p.y / f$.
<code>X::divide(float f, const P& p)</code>	P	Returns an object r.	<i>Postconditions:</i> $r.x == f / p.x$ and $r.y == f / p.y$.
<code>X::equal(const P& l, const P& r)</code>	bool	Returns $l.x == r.x \ \&\& \ l.y == r.y$	

Table 6 — GraphicsMath requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::not_equal(const P& l, const P& r)</code>	bool	Equivalent to return <code>!equal(l, r)</code> ;	
<code>X::negate(const P& p)</code>	P	Returns <code>r</code> , where <code>r.x == -p.x</code> and <code>r.y == -p.y</code>	
<code>X::create_matrix_2d()</code>	M	Equivalent to return <code>create_matrix_2d(1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f)</code> ;	
<code>X::create_matrix_2d(float m00, float m01, float m10, float m11, float m20, float m21)</code>	M	Returns an object <code>m</code>	<i>Postconditions:</i> <code>m.m00 == m00</code> , <code>m.m01 == m01</code> , <code>m.m02 == 0.0f</code> <code>m.m10 == m10</code> , <code>m.m11 == m11</code> , <code>m.m12 == 0.0f</code> <code>m.m20 == m20</code> , <code>m.m21 == m21</code> , <code>m.m22 == 1.0f</code>
<code>X::create_translate(const P& p)</code>	M	Equivalent to return <code>X::create_matrix_2d(1.0f, 0.0f, 0.0f, 1.0f, p.x, p.y)</code> ;	
<code>X::create_scale(const P& p)</code>	M	Equivalent to return <code>X::create_matrix_2d(p.x, 0.0f, 0.0f, p.y, 0.0f, 0.0f)</code> ;	
<code>X::create_rotate(float r)</code>	M	Equivalent to return <code>X::create_matrix_2d(cos(r), sin(r), sin(r), -cos(r), 0.0f, 0.0f)</code> ;	
<code>X::create_rotate(float r, const P& p)</code>	M	Equivalent to return <code>multiply(multiply(create_translate(p), create_rotate(r)), create_translate(negate(p)))</code> ;	
<code>X::create_reflect(float r)</code>	M	Equivalent to return <code>X::create_matrix_2d(cos(r * 2.0f), sin(r * 2.0f), sin(r * 2.0f), -cos(r * 2.0f), 0.0f, 0.0f)</code> ;	
<code>X::create_shear_x(float f)</code>	M	Equivalent to return <code>create_matrix_2d(1.0f, 0.0f, f, 1.0f, 0.0f, 0.0f)</code> ;	
<code>X::create_shear_y(float f)</code>	M	Equivalent to return <code>create_matrix_2d(1.0f, f, 0.0f, 1.0f, 0.0f, 0.0f)</code> ;	
<code>X::m00(M& m, float v)</code>	void		<i>Postconditions:</i> <code>m.m00 == v</code> .
<code>X::m01(M& m, float v)</code>	void		<i>Postconditions:</i> <code>m.m01 == v</code> .
<code>X::m10(M& m, float v)</code>	void		<i>Postconditions:</i> <code>m.m10 == v</code> .

Table 6 — GraphicsMath requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::m11(M& m, float v)	void		<i>Postconditions:</i> m.m11 == v.
X::m20(M& m, float v)	void		<i>Postconditions:</i> m.m20 == v.
X::m21(M& m, float v)	void		<i>Postconditions:</i> m.m21 == v.
X::m00(const M& m)	float	<i>Returns:</i> m.m00	
X::m01(const M& m)	float	<i>Returns:</i> m.m01	
X::m10(const M& m)	float	<i>Returns:</i> m.m10	
X::m11(const M& m)	float	<i>Returns:</i> m.m11	
X::m20(const M& m)	float	<i>Returns:</i> m.m20	
X::m21(const M& m)	float	<i>Returns:</i> m.m21	
X::multiply(const M& l, const M& r)	M	Equivalent to return create_matrix_2d(l.m00 * r.m00 + l.m01 * r.m10, l.m00 * r.m01 + l.m01 * r.m11, l.m10 * r.m00 + l.m11 * r.m10, l.m10 * r.m01 + l.m11 * r.m11, l.m20 * r.m00 + l.m21 * r.m10 + r.m20, l.m20 * r.m01 + l.m21 * r.m11 + r.m21);	
X::translate(M& m, const P& p)	void	Equivalent to m = multiply(m, create_translate(p));	
X::scale(M& m, const P& p)	void	Equivalent to m = multiply(m, create_scale(p));	
X::rotate(M& m, float r)	void	Equivalent to m = multiply(m, create_rotate(r));	
X::rotate(M& m, float r, const P& p)	void	Equivalent to m = multiply(m, create_rotate(r, p));	
X::reflect(M& m, float r)	void	Equivalent to m = multiply(m, create_reflect(r));	
X::shear_x(M& m, float f)	void	Equivalent to m = multiply(m, create_shear_x(f));	
X::shear_y(M& m, float f)	void	Equivalent to m = multiply(m, create_shear_y(f));	
X::is_finite(const M& m)	bool	Equivalent to return isfinite(m.m00) && isfinite(m.m01) && isfinite(m.m10) && isfinite(m.m10) && isfinite(m.m11) && isfinite(m.m20) && isfinite(m.m21);	
X::is_invertible(const M& m)	bool	Equivalent to return (m.m00 * m.m11 - m.m01 * m.m10) != 0.0f;	
X::determinant(const M& m)	float	Equivalent to return m.m00 * m.m11 - m.m01 * m.m10;	

Table 6 — GraphicsMath requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::inverse(const M& m)</code>	M	Equivalent to: <code>float id = 1.0f / determinant(m); return create_matrix_2d((m.m11 * 1.0f - 0.0f * m.m21) * id, -(m.m01 * 1.0f - 0.0f * m.m21) * id, -(m.m10 * 1.0f - 0.0f * m.m20) * id, (m.m00 * 1.0f - 0.0f * m.m20) * id, (m.m10 * m.m21 - m.m11 * m.m20) * id, -(m.m00 * m.m21 - m.m01 * m.m20) * id)</code>	
<code>X::transform_pt(const M& m, const P& p)</code>	P	<code>return create_point_2d(m.m00 * p.x + m10 * p.y + m.m20, m.m01 * p.x + m.m11 * p.y + m.m21);</code>	
<code>X::equal(const M& l, const M& r)</code>	bool	Returns <code>l.m00 == r.m00 && l.m01 == r.m01 && l.m11 == r.m11 && l.m20 == r.m20 && l.m21 == r.m21</code> .	
<code>X::not_equal(const M& l, const M& r)</code>	bool	Equivalent to <code>return !equal(l, r);</code>	
<code>create_display_point()</code>	D	Equivalent to <code>return create_display_point(0, 0);</code>	
<code>create_display_point(int x, int y)</code>	D	Returns an object <code>d</code> .	<i>Postconditions:</i> <code>d.x == x</code> and <code>d.y == y</code> .
<code>X::x(D& d, int x)</code>	void		<i>Postconditions:</i> <code>d.x == x</code> .
<code>X::y(D& d, int y)</code>	void		<i>Postconditions:</i> <code>d.y == y</code> .
<code>X::x(const D&d)</code>	int	Returns <code>d.x</code> .	
<code>X::y(const D&d)</code>	int	Returns <code>d.y</code> .	
<code>X::equal(const D& l, const D& r)</code>	bool	Returns <code>l.x == r.x && l.y == r.y</code> .	
<code>X::not_equal(const D& l, const D& r)</code>	bool	Equivalent to <code>return !equal(l, r);</code>	
<code>X::create_bounding_box()</code>	B	Equivalent to <code>return create_bounding_box(0.0f, 0.0f, 0.0f, 0.0f);</code>	
<code>X::create_bounding_box(float x, float y, float w, float h)</code>	B	Returns an object <code>b</code> .	<i>Postconditions:</i> <code>b.x == x</code> , <code>b.y == y</code> , <code>b.w == w</code> , and <code>b.h == h</code> .
<code>X::create_bounding_box(const P& tl, const P& br)</code>	B	Equivalent to <code>return create_bounding_box(tl.x, tl.y, max(0.0f, br.x - tl.x), max(0.0f, br.y - tl.y));</code>	
<code>X::x(B& b, float x)</code>	void		<i>Postconditions:</i> <code>b.x == x</code> .

Table 6 — GraphicsMath requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::y(B& b, float y)</code>	void		<i>Postconditions:</i> <code>b.y == y.</code>
<code>X::width(B& b, float w)</code>	void		<i>Postconditions:</i> <code>b.w == w.</code>
<code>X::height(B& b, float h)</code>	void		<i>Postconditions:</i> <code>b.h == h.</code>
<code>X::top_left(B& b, const P& p)</code>	void		<i>Postconditions:</i> <code>b.x == p.x</code> and <code>b.y == p.y.</code>
<code>X::bottom_right(B& b, const P& p)</code>	void		<i>Postconditions:</i> <code>b.w == max(p.x - b.x, 0.0f)</code> and <code>b.h == max(p.y - b.y, 0.0f).</code>
<code>X::x(const B& b)</code>	float	Returns <code>b.x.</code>	
<code>X::y(const B& b)</code>	float	Returns <code>b.y.</code>	
<code>X::width(const B& b)</code>	float	Returns <code>b.w.</code>	
<code>X::height(const B& b)</code>	float	Returns <code>b.h.</code>	
<code>X::top_left(const B& b)</code>	P	Returns an object <code>p.</code>	<i>Postconditions:</i> <code>p.x == b.x</code> and <code>p.y == b.y.</code>
<code>X::bottom_right(const B& b)</code>	P	Returns an object <code>p.</code>	<i>Postconditions:</i> <code>p.x == b.x + b.w</code> and <code>p.y == b.y + b.h.</code>
<code>X::equal(const B& l, const B& r)</code>	bool	Returns <code>l.x == r.x && l.y == r.y && l.w == r.w && l.h == r.h.</code>	
<code>X::not_equal(const B& l, const B& r)</code>	bool	Equivalent to return <code>!equal(l, r);</code>	
<code>X::create_circle()</code>	C	Equivalent to return <code>create_circle(create_point_2d(0.0f, 0.0f), 0.0f);</code>	
<code>X::create_circle(const P& p, float r)</code>	C	Returns an object <code>c.</code>	<i>Requires:</i> <code>r >= 0.0f.</code> <i>Postconditions:</i> <code>c.x == p.x,</code> <code>c.y == p.y,</code> and <code>c.r == r.</code>
<code>X::center(C& c, const P& p)</code>	void		<i>Postconditions:</i> <code>c.x == p.x</code> and <code>c.y == p.y.</code>
<code>X::radius(C& c, float r)</code>	void		<i>Requires:</i> <code>r >= 0.0f.</code> <i>Postconditions:</i> <code>c.r == r.</code>

Table 6 — GraphicsMath requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::center(const C& c)</code>	P	Returns an object p.	<i>Postconditions:</i> p.x == c.x and p.y == c.y.
<code>X::radius(const C& c)</code>	float	Returns c.r.	
<code>X::equal(const C& l, const C& r)</code>	bool	Returns l.x == r.x && l.y == r.y && l.r == r.r.	
<code>X::not_equal(const C& l, const C& r)</code>	bool	Equivalent to return !equal(l, r);	
<code>X::point_for_angle(float a, float m)</code>	P	Returns transform_pt(create_rotate(a), create_point_2d(m, 0.0f)).	
<code>X::point_for_angle(float a, const P& r)</code>	P	Returns multiply(transform_pt(create_rotate(a), create_point_2d(1.0f, 0.0f)), r).	
<code>X::angle_for_point(const P& c, const P& p)</code>	float	Equivalent to: const float co = pi<float> / 180'000.0f; auto a = atan2(-(p.y - c.y), p.x - c.x); if (abs(a) < co abs(a - two_pi<float>) < co) { return 0.0f; } if (a < 0.0f) { return a + two_pi<float>; } return a;	
<code>X::arc_start(const P& c, float sa, const P& r, const M& m)</code>	P	Equivalent to: auto lm = m; lm.m20 = 0.0f; lm.m21 = 0.0f; return add(c, transform_pt(lm, point_for_angle(sa, r)));	
<code>X::arc_center(const P& c, float sa, const P& r, const M& m)</code>	P	Equivalent to: auto lm = m; lm.m20 = 0.0f; lm.m21 = 0.0f; auto o = point_for_angle(two_pi<float> - sa, r); o.y = -o.y; return subtract(c, transform_pt(lm, o));	
<code>X::arc_end(const P& c, float ea, const P& r, const M& m)</code>	P	Equivalent to: auto lm = m; lm.m20 = 0.0f; lm.m21 = 0.0f; auto pt = transform_pt(create_rotate(ea), r); pt.y = -pt.y; return add(c, transform_pt(lm, pt));	

9 Graphics surfaces

[io2d.graphsurf]

9.1 General

[io2d.graphsurf.general]

- ¹ This Clause defines requirements on *GraphicsSurfaces* types.
- ² Most classes specified in [Clause 13](#), [Clause 14](#), and [Clause 16](#) need a set of related types and functions to complete the definition of their semantics. These types and functions are provided as a set of *typedef-names* and nested classes containing *typedef-names* and `static` member functions in the template argument `GraphicsSurfaces` used by each such template. This Clause defines the names of the classes and the semantics of their members.
- ³ [*Note*: It is important to remember that in C++, the requirements are not to execute each expression at the time and place that it occurs in the program but instead is to emulate the observable behavior of the abstract machine described by the C++ standard, i.e. to follow the as-if rule.
- ⁴ There are only a few 2D graphics operations that produce observable behavior, those being the operations that are visible to the user either by displaying the results of graphics operations to the output surface or by saving the results to a file. Thus, most operations can be implemented in whatever manner best suits the environment that the *GraphicsSurfaces* types are designed to target so long as the end result complies with the as-if rule.
- ⁵ For example, hardware accelerated implementations could record the rendering and composing operations, creating vertex buffers, index buffers, and state objects at appropriate times and adding them to a command list, and then submit that list to the hardware when observable behavior occurs in order to achieve optimal performance. This is a highly simplified description that is meant to provide some indication as to how such implementations could work since the details of hardware accelerated optimization of graphics operations are quite complex and are beyond the scope of this Technical Specification . — *end note*]

9.2 Requirements

[io2d.graphsurf.reqs]

9.2.1 Classes

[io2d.graphsurf.reqs.classes]

- ¹ A *GraphicsSurfaces* type is a class template with one type parameter. The template type argument of an instantiation of a *GraphicsSurfaces* specialization shall meet the requirements of a *GraphicsMath* type (See: [Clause 8](#)).
- ² A *GraphicsSurfaces* type contains a *typedef-name* `graphics_math_type`, which is an identifier for the template argument. It also contains a *typedef-name* `graphics_surfaces_type`, which is an identifier for the *GraphicsSurfaces* type.
- ³ [*Example*:


```
template <class GraphMath>
struct GraphSurf {
    using graphics_math_type = GraphMath;
    using graphics_surfaces_type = GraphSurf;
    // ...
};
```

 — *end example*]
- ⁴ A *GraphicsSurfaces* is required to have the following public nested classes:
 1. `additional_image_file_formats`
 2. `additional_formats`
 3. `brushes`
 4. `paths`
 5. `surface_states`
 6. `surfaces`

9.2.2 `additional_image_file_formats` requirements [io2d.graphsurf.reqs.addingform]

- 1 Let `X` be a `GraphicsSurfaces` type.
- 2 The `X::additional_image_file_formats` class contains zero or more `image_file_format` enumerators that represent implementation-defined additional data formats that the implementation can both construct an `image_surface` object from using the appropriate constructor and save an `image_surface` object to using `image_surface::save`. These are called *read/write image format enumerators*.
- 3 The values of read/write image format enumerators shall be in the range [10000, 19999].
- 4 The `X::additional_image_file_formats` class also contains the following nested classes:
 1. `read_only`
 2. `write_only`
- 5 The `additional_image_file_formats` class contains zero or more `image_file_format` enumerators that represent implementation-defined additional data formats that the implementation can construct an `image_surface` object from using the appropriate constructor but cannot save an `image_surface` object to using `image_surface::save`. These are called *read only image format enumerators*.
- 6 The values of read only image format enumerators shall be in the range [20000, 29999].
- 7 The `additional_image_file_formats::write_only` class contains zero or more `image_file_format` enumerators that represent implementation-defined additional data formats that the implementation can construct an `image_surface` object from using the appropriate constructor and save an `image_surface` object to using `image_surface::save` but cannot construct an `image_surface` object from using any constructor. These are called *write only image format enumerators*.
- 8 The values of write only image format enumerators shall be in the range [30000, 39999].

9.2.3 `additional_formats` requirements [io2d.graphsurf.reqs.addform]

- 1 Let `X` be a `GraphicsSurfaces` type.
- 2 The `X::additional_formats` class contains zero or more `format` enumerators that represent implementation-defined additional visual data formats that the implementation supports.
- 3 The size in bytes, byte order, and interpretation of values within each channel of each additional visual data format is *implementation-defined*.
- 4 The values of the additional visual data format enumerators shall be in the range [10000, 39999].

9.2.4 `brushes` requirements [io2d.graphsurf.reqs.brushes]

- 1 Brushes are described in [Clause 14](#).
- 2 Let `X` be a `GraphicsSurfaces` type.
- 3 Let `M` be `X::graphics_math_type`.
- 4 [Table 7](#) describes the observable effects of a member functions of `X::brushes`.
- 5 `X::brushes` contains a *typedef-name*, `brush_data_type`, which is an identifier for a class type capable of storing all data required to support a brush of any type described in [Clause 14](#). [*Note: The information in [14.7.3](#) is particularly important. — end note*]

Table 7 — X::brushes requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::brushes::create_brush(const rgba_color& c)	brush_data_type	Returns an object b.	<i>Postconditions:</i> b is a solid color brush, its visual data is c (See: 14.7.3.1), and X::brushes::get_brush_type(b) == brush_type::solid_color. [<i>Note:</i> Solid color does not imply opaque. The color may be translucent or even transparent. — <i>end note</i>]
template <class InputIterator> create_brush(const basic_point_2d<M>& be, const basic_point_2d<M>& en, InputIterator first, InputIterator last)	brush_data_type	Returns an object b.	<i>Postconditions:</i> b is a linear gradient brush, its begin point is be, its end point is en, its gradient stops are formed using the sequential series of gradient_stop objects beginning at first and ending at last - 1 (See 14.2.2 and 14.2.4), and X::brushes::get_brush_type(b) == brush_type::linear.

Table 7 — X::brushes requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<pre>create_brush(const basic_point_2d<M>& be, const basic_point_2d<M>& en, initializer_- list<gradient_stop> il)</pre>	brush_data_type	Returns an object b.	<i>Postconditions:</i> b is a linear gradient brush, its begin point is be, its end point is en, its gradient stops are formed using the sequential series of gradient_stop objects in il (See 14.2.2 and 14.2.4), and X::brushes::get_brush_type(b) == brush_type::linear.
<pre>template <class InputIterator> create_brush(const basic_circle<M>& be, const basic_circle<M>& en, InputIterator first, InputIterator last)</pre>	brush_data_type	Returns an object b.	<i>Postconditions:</i> b is a radial gradient brush, its start circle is be, its end circle is en, its gradient stops are formed using the sequential series of gradient_stop objects beginning at first and ending at last - 1 (See 14.2.3 and 14.2.4), and X::brushes::get_brush_type(b) == brush_type::radial.

Table 7 — X::brushes requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<pre>create_brush(const basic_circle<M>& be, const basic_circle<M>& en, initializer_- list<gradient_stop> il)</pre>	brush_data_type	Returns an object b.	<i>Postconditions:</i> b is a radial gradient brush, its start circle is be, its end circle is en, its gradient stops are formed using the sequential series of gradient_stop objects in il (See 14.2.3 and 14.2.4), and X::brushes::get_brush_type(b) == brush_type::radial.
<pre>create_brush(basic_- image_- surface<graphics_- surfaces_type>&& i)</pre>	brush_data_type	Returns an object b.	<i>Postconditions:</i> b is a surface brush, its visual data is the raster graphics data from i, and X::brushes::get_brush_type(b) == brush_type::surface
<pre>get_brush_- type(const brush_data_type& data)</pre>	brush_type	Returns the brush type of data.	

9.2.5 paths requirements

[io2d.graphsurf.reqs.paths]

- ¹ Paths are described in [Clause 13](#).
- ² Let X be a GraphicsSurfaces type.
- ³ Let G be X::graphics_math_type.
- ⁴ Table 10 describes the observable effects of the member functions of X::paths.
- ⁵ Table 8 defines the required *typedef-names* in X::paths, which are identifiers for class types capable of storing all data required to support the corresponding class template.

Table 8 — X::paths typedef-names

<i>typedef-name</i>	Class data
abs_cubic_curve_data_type	basic_figure_items<X>::abs_cubic_curve
abs_line_data_type	basic_figure_items<X>::abs_line
abs_matrix_data_type	basic_figure_items<X>::abs_matrix
abs_new_figure_data_type	basic_figure_items<X>::abs_new_figure
abs_quadratic_curve_data_type	basic_figure_items<X>::abs_quadratic_curve
arc_data_type	basic_figure_items<X>::arc
close_figure_data_type	basic_figure_items<X>::close_data

Table 8 — X::paths typedef-names (continued)

<i>typedef-name</i>	Class template
<code>interpreted_path_data_type</code>	<code>basic_interpreted_path<X></code>
<code>rel_cubic_curve_data_type</code>	<code>basic_figure_items<X>::rel_cubic_curve</code>
<code>rel_line_data_type</code>	<code>basic_figure_items<X>::rel_line</code>
<code>rel_matrix_data_type</code>	<code>basic_figure_items<X>::rel_matrix</code>
<code>rel_new_figure_data_type</code>	<code>basic_figure_items<X>::rel_new_figure</code>
<code>rel_quadratic_curve_data_type</code>	<code>basic_figure_items<X>::rel_quadratic_curve</code>
<code>revert_matrix_data_type</code>	<code>basic_figure_items<X>::revert_matrix</code>

- ⁶ [*Note:* An object of type `basic_interpreted_path<X>` is an immutable object. As such, the contents of the class type for which `X::paths::interpreted_path_data_type` is an identifier are able to be highly tailored to the platform and environment targeted by X. — *end note*]
- ⁷ In Table 10, AC denotes the type `X::paths::abs_cubic_curve_data_type`, AL denotes the type `X::paths::abs_line_data_type`, AM denotes the type `X::paths::abs_matrix_data_type`, AN denotes the type `X::paths::abs_new_figure_data_type`, AQ denotes the type `X::paths::abs_quadratic_curve_data_type`, ARC denotes the type `X::paths::arc_data_type`, BB denotes the type `basic_bounding_box<G>`, FI denotes the type `basic_figure_items<X>::figure_item`, IP denotes the type `X::paths::interpreted_path_data_type`, RC denotes the type `X::paths::rel_cubic_curve_data_type`, RL denotes the type `X::paths::rel_line_data_type`, RM denotes the type `X::paths::rel_matrix_data_type`, RN denotes the type `X::paths::rel_new_figure_data_type`, RQ denotes the type `X::paths::rel_quadratic_curve_data_type`, M denotes the type `basic_matrix_2d<G>`, and P denotes the type `basic_point_2d<G>`.
- ⁸ In order to describe the observable effects of functions contained in Table 10, Table 9 describes the types contained in X as-if they possessed certain member data.
- ⁹ [*Note:* Certain types do not require any member data to describe the observable effects of the functions they are used by and thus do not appear in Table (9). — *end note*]

Table 9 — X::paths type member data

Type	Member data	Member data type
<code>abs_cubic_curve_data_type</code>	<code>cpt1</code>	<code>basic_point_2d<G></code>
<code>abs_cubic_curve_data_type</code>	<code>cpt2</code>	<code>basic_point_2d<G></code>
<code>abs_cubic_curve_data_type</code>	<code>ept</code>	<code>basic_point_2d<G></code>
<code>abs_line_data_type</code>	<code>pt</code>	<code>basic_point_2d<G></code>
<code>abs_matrix_data_type</code>	<code>m</code>	<code>basic_matrix_2d<G></code>
<code>abs_new_figure_data_type</code>	<code>pt</code>	<code>basic_point_2d<G></code>
<code>abs_quadratic_curve_data_type</code>	<code>cpt</code>	<code>basic_point_2d<G></code>
<code>abs_quadratic_curve_data_type</code>	<code>ept</code>	<code>basic_point_2d<G></code>
<code>arc_data_type</code>	<code>rad</code>	<code>basic_point_2d<G></code>
<code>arc_data_type</code>	<code>rot</code>	<code>float</code>
<code>arc_data_type</code>	<code>sa</code>	<code>float</code>
<code>rel_cubic_curve_data_type</code>	<code>cpt1</code>	<code>basic_point_2d<G></code>
<code>rel_cubic_curve_data_type</code>	<code>cpt2</code>	<code>basic_point_2d<G></code>
<code>rel_cubic_curve_data_type</code>	<code>ept</code>	<code>basic_point_2d<G></code>
<code>rel_line_data_type</code>	<code>pt</code>	<code>basic_point_2d<G></code>
<code>rel_matrix_data_type</code>	<code>m</code>	<code>basic_matrix_2d<G></code>
<code>rel_new_figure_data_type</code>	<code>pt</code>	<code>basic_point_2d<G></code>
<code>rel_quadratic_curve_data_type</code>	<code>cpt</code>	<code>basic_point_2d<G></code>
<code>rel_quadratic_curve_data_type</code>	<code>ept</code>	<code>basic_point_2d<G></code>

Table 10 — X::paths requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::paths::create_-\nabs_cubic_curve()	AC	<i>Effects:</i> Equivalent to return\ncreate_abs_cubic_curve(P(),\nP(), P());	
X::paths::create_-\nabs_cubic_-\ncurve(const P& cpt1,\nconst P& cpt2,\nconst P& ept)	abs_cubic_curve_-\ndata_type	<i>Returns:</i> An object ac.	<i>Postconditions:</i> \nac.cpt1 ==\ncpt1, ac.cpt2\n== cpt2, and\nac.ept ==\nept.
X::paths::control_-\npt1(AC& ac, const\nP& pt)	void		<i>Postconditions:</i> \nac.cpt1 ==\npt.
X::paths::control_-\npt2(AC& ac, const\nP& pt)	void		<i>Postconditions:</i> \nac.cpt2 ==\npt.
X::paths::end_-\npt(AC& ac, const P&\npt)	void		<i>Postconditions:</i> \nac.ept == pt.
X::paths::control_-\npt1(const AC&\nac)	P	<i>Returns:</i> ac.cpt1.	
X::paths::control_-\npt2(const AC&\nac)	P	<i>Returns:</i> ac.cpt2.	
X::paths::end_-\npt(const AC&\nac)	P	<i>Returns:</i> ac.ept.	
X::paths::create_-\nabs_line()	AL	<i>Effects:</i> Equivalent to return\ncreate_abs_line(P);	
X::paths::create_-\nabs_line(const P&\np)	AL	<i>Returns:</i> An object al.	<i>Postconditions:</i> \nal.pt == p.
X::paths::to(AL& al,\nconst P& p)	void		<i>Postconditions:</i> \nal.pt == p.
X::paths::to(const\nAL& al)	P	<i>Returns:</i> al.pt.	
X::paths::create_-\nabs_matrix()	AM	Equivalent to return\ncreate_abs_matrix(M());	
X::paths::create_-\nabs_matrix(const M&\nm)	AM	<i>Returns:</i> An object am.	<i>Postconditions:</i> \nam.m == m.
X::paths::matrix(AM&\nam, const M& m)	void		<i>Postconditions:</i> \nam.m == m.
X::paths::matrix(constM\nAM& am)		<i>Returns:</i> am.m.	
X::paths::create_-\nabs_new_figure()	AN	<i>Effects:</i> Equivalent to return\ncreate_abs_new_-\nfigure(P());	
X::paths::create_-\nabs_new_-\nfigure(const P&\np)	AN	<i>Returns:</i> An object an.	<i>Postconditions:</i> \nan.pt == p.
X::paths::at(AN& an,\nconst P& p)	void		<i>Postconditions:</i> \nan.pt == p.

Table 10 — X::paths requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::paths::at(const AN& an)	P	Returns: an.pt.	
X::paths::create_-abs_quadratic_curve()	AQ	Equivalent to return create_-abs_quadratic_curve(P(), P());	
X::paths::create_-abs_quadratic_curve(const P& cpt, const P& ept)	AQ	Returns: An object aq.	Postconditions: aq.cpt == cpt and aq.ept == ept.
X::paths::control_pt(AQ& aq, const P& p)	void		Postconditions: aq.cpt == p.
X::paths::end_pt(AQ& aq, const P& p)	void		Postconditions: aq.ept == p.
X::paths::control_pt(const AQ& aq)	P	Returns: aq.cpt.	
X::paths::end_pt(const AQ& aq)	P	Returns: aq.ept.	
X::paths::create_arc()	ARC	Effects: Equivalent to return create_arc(P(), 0.0f, 0.0f);	
X::paths::create_arc(const P& rad, float rot, float sa)	ARC	Returns: An object arc.	Postconditions: arc.rad == rad, arc.rot == rot, and arc.sa == sa.
X::paths::radius(ARC& arc, const P& rad)	void		Postconditions: arc.rad == rad.
X::paths::rotation(ARC& arc, float rot)	void		Postconditions: arc.rot == rot.
X::paths::start_angle(ARC& arc, float sa)	void		Postconditions: arc.sa == sa.
X::paths::radius(const P ARC& arc)	float	Returns: arc.rad.	
X::paths::rotation(const P ARC& arc)	float	Returns: arc.rot.	
X::paths::start_angle(const P ARC& arc)	float	Returns: arc.sa.	

Table 10 — X::paths requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::paths::center(constP ARC& arc, const P& spt, const M& m)		<i>Returns:</i> As-if: <pre> auto lmtx = m; lmtx.m20(0.0f); lmtx.m21(0.0f); auto ctrOffset = point_for_angle<G>(two_pi<float> - arc.sa, arc.rad); ctrOffset.y(-ctrOffset.y); return spt - ctrOffset * lmtx; </pre>	[<i>Note:</i> spt is the starting point of the arc. m is the transformation matrix being used. — <i>end note</i>]
X::paths::end_pt(const ARC& arc, const P& spt, const M& m)	P	<i>Returns:</i> As-if: <pre> auto lmtx = m; lmtx.m20(0.0f); lmtx.m21(0.0f); auto tfrm = M::create_rotate(arc.sa + arc.rot); auto pt = arc.rad * tfrm; pt.y(-pt.y()); return spt + pt * lmtx; </pre>	[<i>Note:</i> spt is the starting point of the arc. m is the transformation matrix being used. — <i>end note</i>]
X::paths::create_interpreted_path()	IP	<i>Returns:</i> An object ip.	<i>Postconditions:</i> ip has zero figures (See: 13.3.16)

Table 10 — X::paths requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::paths::create_interpreted_path(const BB& bb)	IP	<pre> <i>Effects:</i> Equivalent to: return create_interpreted_path(figureItem(in_place_ type<typename basic_figure_ items<graphics_surfaces_ type>::abs_new_figure>, bb.top_left()), figureItem(in_place_ type<typename basic_figure_ items<graphics_surfaces_ type>::rel_line>, basic_point_ 2d<GraphicsMath>(bb.width(), 0.0f)), figureItem(in_ place_type<typename basic_figure_ items<graphics_surfaces_ type>::rel_line>, basic_point_ 2d<GraphicsMath>(0.0f, bb.height())), figureItem(in_place_ type<typename basic_figure_ items<graphics_surfaces_ type>::rel_line>, basic_point_ 2d<GraphicsMath>(-bb.width(), 0.0f)), figureItem(in_ place_type<typename basic_figure_ items<graphics_surfaces_ type>::close_figure>); </pre>	
X::paths::create_interpreted_path(initializer_list<FI> il)	IP	<pre> <i>Effects:</i> Equivalent to: return create_interpreted_ path(begin(il), end(il)); </pre>	

Table 10 — X::paths requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
template <class ForwardIterator> X::paths::create_interpreted_path<ForwardIterator first, ForwardIterator last	IP	<i>Returns:</i> An object ip.	<i>Postconditions:</i> ip contains a zero or more figure items as determined by evaluating the sequence of FI objects beginning with first and ending with last in the manner described in 13.3.16. <i>Remarks:</i> The internal data of the interpreted path should be in a form that is best suited to take advantage of the platform and environment targeted by X.
X::paths::create_rel_cubic_curve()	RC	<i>Effects:</i> Equivalent to return create_rel_cubic_curve(P(), P(), P());	
X::paths::create_rel_cubic_curve(const P& cpt1, const P& cpt2, const P& ept)	RC	<i>Returns:</i> An object rc.	<i>Postconditions:</i> rc.cpt1 == cpt1, rc.cpt2 == cpt2, and rc.ept == ept.
X::paths::control_pt1(RC& rc, const P& pt)	void		<i>Postconditions:</i> rc.cpt1 == pt.
X::paths::control_pt2(RC& rc, const P& pt)	void		<i>Postconditions:</i> rc.cpt2 == pt.
X::paths::end_pt(RC& rc, const P& pt)	void		<i>Postconditions:</i> rc.ept == pt.
X::paths::control_pt1(const RC& a)	P	<i>Returns:</i> rc.cpt1.	
X::paths::control_pt2(const RC& rc)	P	<i>Returns:</i> rc.cpt2.	
X::paths::end_pt(const RC& rc)	P	<i>Returns:</i> rc.ept.	

Table 10 — X::paths requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::paths::create_rel_line()	RL	<i>Effects:</i> Equivalent to return <code>create_rel_line(P)</code> ;	
X::paths::create_rel_line(const P& p)	RL	<i>Returns:</i> An object <code>rl</code> .	<i>Postconditions:</i> <code>rl.pt == p</code> .
X::paths::to(RL& al, const P& p)	void		<i>Postconditions:</i> <code>rl.pt == p</code> .
X::paths::to(const RL& rl)	P	<i>Returns:</i> <code>rl.pt</code> .	
X::paths::create_rel_matrix()	RM	Equivalent to return <code>create_rel_matrix(M())</code> ;	
X::paths::create_rel_matrix(const M& m)	RM	<i>Returns:</i> An object <code>rm</code> .	<i>Postconditions:</i> <code>rm.m == m</code> .
X::paths::matrix(RM& am, const M& m)	void		<i>Postconditions:</i> <code>rm.m == m</code> .
X::paths::matrix(const RM& rm)		<i>Returns:</i> <code>am.m</code> .	
X::paths::create_rel_new_figure()	RN	<i>Effects:</i> Equivalent to return <code>create_rel_new_figure(P())</code> ;	
X::paths::create_rel_new_figure(const P& p)	RN	<i>Returns:</i> An object <code>rn</code> .	<i>Postconditions:</i> <code>rn.pt == p</code> .
X::paths::at(RN& rn, const P& p)	void		<i>Postconditions:</i> <code>rn.pt == p</code> .
X::paths::at(const RN& rn)	P	<i>Returns:</i> <code>rn.pt</code> .	
X::paths::create_rel_quadratic_curve()	RQ	Equivalent to return <code>create_rel_quadratic_curve(P(), P())</code> ;	
X::paths::create_rel_quadratic_curve(const P& cpt, const P& ept)	RQ	<i>Returns:</i> An object <code>rq</code> .	<i>Postconditions:</i> <code>rq.cpt == cpt</code> and <code>rq.ept == ept</code> .
X::paths::control_pt(RQ& rq, const P& p)	void		<i>Postconditions:</i> <code>rq.cpt == p</code> .
X::paths::end_pt(RQ& rq, const P& p)	void		<i>Postconditions:</i> <code>rq.ept == p</code> .
X::paths::control_pt(const RQ& rq)	P	<i>Returns:</i> <code>rq.cpt</code> .	
X::paths::end_pt(const RQ& rq)	P	<i>Returns:</i> <code>rq.ept</code> .	

9.2.6 surface_state_props requirements

[io2d.graphsurf.reqs.surfstate]

- ¹ Surface state data are described in [Clause 15](#).
- ² Let X be a GraphicsSurfaces type.
- ³ Let G be X::graphics_math_type.

- ⁴ Table 13 describes the observable effects of the member functions of `X::surface_state_props`.
- ⁵ Table 11 defines the required *typedef-names* in `X::surface_state_props`, which are identifiers for class types capable of storing all data required to support the corresponding class template.

Table 11 — `X::surface_state_props` typedef-names

<i>typedef-name</i>	Class data
<code>render_props_data_type</code>	<code>basic_render_props</code>
<code>brush_props_data_type</code>	<code>basic_brush_props</code>
<code>clip_props_data_type</code>	<code>basic_clip_props</code>
<code>stroke_props_data_type</code>	<code>basic_stroke_props</code>
<code>mask_props_data_type</code>	<code>basic_mask_props</code>
<code>dashes_data_type</code>	<code>basic_dashes</code>

- ⁶ In Table 13, RE denotes the type `X::surface_state_props::render_props_data_type`, BR denotes the type `X::surface_state_props::brush_props_data_type`, CL denotes the type `X::surface_state_props::clip_props_data_type`, ST denotes the type `X::surface_state_props::stroke_props_data_type`, FP denotes the type `X::surface_state_props::fill_props_data_type`, MA denotes the type `X::surface_state_props::mask_props_data_type`, DA denotes the type `X::surface_state_props::dashes_data_type`, BB denotes the type `basic_bounding_box<G>`, IP denotes the type `basic_interpreted_path<X>`, FI denotes the type `basic_figure_items<X>::figure_item`, M denotes the type `basic_matrix_2d<G>`, and P denotes the type `basic_point_2d<G>`.
- ⁷ In order to describe the observable effects of functions contained in Table 13, Table 12 describes the types contained in X as-if they possessed certain member data.

Table 12 — `X::surface_state_props` type member data

Type	Member data	Member data type
<code>render_props_data_type</code>	fi	filter
<code>render_props_data_type</code>	m	M
<code>render_props_data_type</code>	c	<code>compositing_op</code>
<code>brush_props_data_type</code>	w	<code>wrap_mode</code>
<code>brush_props_data_type</code>	fi	filter
<code>brush_props_data_type</code>	m	M
<code>clip_props_data_type</code>	<code>optional<IP></code>	c
<code>clip_props_data_type</code>	fr	<code>fill_rule</code>
<code>stroke_props_data_type</code>	lw	float
<code>stroke_props_data_type</code>	ml	float
<code>stroke_props_data_type</code>	lc	<code>line_cap</code>
<code>stroke_props_data_type</code>	lj	<code>line_join</code>
<code>stroke_props_data_type</code>	aa	<code>antialias</code>
<code>fill_props_data_type</code>	fr	<code>fill_rule</code>
<code>fill_props_data_type</code>	aa	<code>antialias</code>
<code>mask_props_data_type</code>	wm	<code>wrap_mode</code>
<code>mask_props_data_type</code>	fi	filter
<code>mask_props_data_type</code>	m	M
<code>dashes_props_data_type</code>	o	float
<code>dashes_props_data_type</code>	p	<code>vector<float></code>

Table 13 — X::surface_state_props requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::surface_state_props::create_render_props(filter f = filter::good, M m = M{}, compositing op c = compositing_op::over)	RE	Returns: An object r.	Postconditions: r.fi == f, r.m == m, and r.c == c.
X::surface_state_props::filtering(RA& r, filter fi)	void		Postconditions: r.fi == fi.
X::surface_state_props::surface_matrix(RA& r, const M& m)	void		Postconditions: ra.m == m.
X::surface_state_props::compositing(RA& r, compositing_op c)	void		Postconditions: ra.c == c.
X::surface_state_props::filtering(const RA& r)	filter	Returns: r.fi.	
X::surface_state_props::surface_matrix(const RA& r)	M	Returns: r.m.	
X::surface_state_props::compositing(const RA& r)	compositing_op	Returns: r.c.	
X::surface_state_props::create_brush_props(wrap_mode w = wrap_mode::none, filter fi = filter::good, const M& m = M{})	BR	Returns: An object b.	Postconditions: b.w == w, b.fi == fi, and b.m == m.
X::surface_state_props::brush_wrap_mode(BR& b, wrap_mode w)	void		Postconditions: b.w == w.
X::surface_state_props::brush_filter(BR& b, filter fi)	void		Postconditions: b.fi == fi.
X::surface_state_props::brush_matrix(BR& b, const M& m)	void		Postconditions: b.m == m.
X::surface_state_props::brush_wrap_mode(const BR& b)	wrap_mode	Returns: b.w.	

Table 13 — X::surface_state_props requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::surface_state_props::brush_filter(const BR& b)	filter	Returns: b.fi.	
X::surface_state_props::brush_matrix(const BR& b)	M	Returns: b.m.	
X::surface_state_props::create_clip_props()	CL	Returns: An object c.	Postconditions: c.c == nullopt and c.fr == fill_ rule::winding.
X::surface_state_props::create_clip_props(const BB& b, fill_rule fr)	CL	Returns: An object c.	Postconditions: c.c == clip(c, b) and c.fr == fr.
template <class Allocator> X::surface_state_props::create_clip_props(const basic_path_builder<X, Allocator>& pb, fill_rule fr)	CL	Returns: An object c.	Postconditions: c.c == IP(pb) and c.fr == fr.
template <class InputIterator> X::surface_state_props::create_clip_props(InputIterator first, InputIterator last, fill_rule fr)	CL	Returns: An object c.	Postconditions: c.c == IP(first, last) and c.fr == fr.
X::surface_state_props::create_clip_props(initializer_list<FI> il, fill_rule fr)	CL	Returns: An object c.	Postconditions: c.c == IP(il) and c.fr == fr.
X::surface_state_props::create_clip_props(const IP& ip, fill_rule fr)	CL	Returns: An object c.	Postconditions: c.c == ip and c.fr == fr.
X::surface_state_props::clip(CL& c, nullopt_t)	void		Postconditions: c.c == nullopt.
X::surface_state_props::clip(CL& c, const BB& b)	void		Postconditions: c.c == IP(b).

Table 13 — X::surface_state_props requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
template <class Allocator> X::surface_state_props::clip(CL& c, const basic_path_builder<X, Allocator>& pb)	void		<i>Postconditions:</i> c.c == IP(pb).
template <class InputIterator> X::surface_state_props::clip(CL& c, InputIterator first, InputIterator last)	void		<i>Postconditions:</i> c.c == IP(first, last).
X::surface_state_props::clip(CL& c, initializer_list<FI> il)	void		<i>Postconditions:</i> c.c == IP(il).
X::surface_state_props::clip(CL& c, const IP& ip)	void		<i>Postconditions:</i> c.c == ip.
X::surface_state_props::clip_fill_rule(CL&c, fill_rule fr)	void		<i>Postconditions:</i> c.fr == fr.
X::surface_state_props::clip(const CL& c)	IP	<i>Returns:</i> c.c.	
X::surface_state_props::clip_fill_rule(const CL& c)	fill_rule	<i>Returns:</i> c.fr.	
X::surface_state_props::create_stroke_props(float lw = 2.0f, line_cap lc = line_cap::none, line_join lj = line_join::miter, float ml = 10.0f, antialias aa = antialias::good)	ST	<i>Returns:</i> An object s.	<i>Requires:</i> lw >= 0.0f, ml >= 1.0f, and ml <= max_miter_limit(). <i>Postconditions:</i> s.lw == lw, s.lc == lc, s.lj == lj, s.ml == ml, and s.aa == aa.
X::surface_state_props::line_width(ST& s, float lw)	void		<i>Postconditions:</i> s.lw == lw.
X::surface_state_props::stroke_line_cap(ST& s, line_cap lc)	void		<i>Postconditions:</i> s.lc == lc.

Table 13 — X::surface_state_props requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::surface_state_ props::stroke_line_ join(ST& s, line_join lj)	void		<i>Postconditions:</i> s.lj == lj.
X::surface_state_ props::miter_ limit(ST& s, float ml)	void		<i>Postconditions:</i> s.ml == ml.
X::surface_state_ props::antialiasing(ST& s, antialias aa)	void	<i>Postconditions:</i> s.aa == aa.	
X::surface_state_ props::line_ width(const ST& s)	float	<i>Returns:</i> s.lw.	
X::surface_state_ props::stroke_line_ cap(const ST& s)	line_cap	<i>Returns:</i> s.lc.	
X::surface_state_ props::stroke_line_ join(const ST& s)	line_join	<i>Returns:</i> s.lj.	
X::surface_state_ props::miter_ limit(const ST& s)	float	<i>Returns:</i> s.ml.	
X::surfaces_state_ props::antialiasing(const ST& s)	antialias	<i>Returns:</i> s.aa.	
X::surface_state_ props::max_miter_ limit()	float	<i>Returns:</i> An implementation-defined maximum value for ST::ml.	
X::surface_state_ props::create_fill_ props(fill_rule fr = fill_rule::winding, antialias aa = antialias::good)	FP	<i>Returns:</i> An object f.	<i>Postconditions:</i> f.fr == fr and f.aa == aa.
X::surface_state_ props::fill_ rule(FP& f, fill_rule fr)	void		<i>Postconditions:</i> f.fr == fr.
X::surface_state_ props::antialiasing(FP& f, antialias aa)	void		<i>Postconditions:</i> f.aa == aa.
X::surface_state_ props::fill_ rule(const FP& f)	fill_rule	<i>Returns:</i> f.fr.	
X::surface_state_ props::antialiasing(const FP& f)	antialias	<i>Returns:</i> f.aa.	

Table 13 — X::surface_state_props requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::surface_state_props::create_mask_props(wrap_mode wm = wrap_mode::none, filter fi = filter::good, const M& m = M())	MA	Returns: An object ma.	Postconditions: ma.wm == wm, ma.fi == fi, and ma.m == m.
X::surface_state_props::mask_wrap_mode(MA& ma, wrap_mode wm)	void		Postconditions: ma.wm == wm.
X::surface_state_props::mask_filter(MA& ma, filter fi)	void		Postconditions: ma.fi == fi.
X::surface_state_props::mask_matrix(MA& ma, const M& m)	void		Postconditions: ma.m == m.
X::surface_state_props::mask_wrap_mode(const MA& ma)	wrap_mode	Returns: ma.wm.	
X::surface_state_props::mask_filter(const MA& ma)	filter	Returns: ma.fi.	
X::surface_state_props::mask_matrix(const MA& ma)	M	Returns: ma.m.	
X::surface_state_props::create_dashes()	DA	Returns: An object d.	Postconditions: d.o == 0.0f and d.p == vector<float>{}
template <class InputIterator> X::surface_state_props::create_dashes(float o, InputIterator first, InputIterator last)	DA	Returns: An object d.	Postconditions: d.o == o and d.p == vector<float>(first, last).
X::surface_state_props::create_dashes(float o, initializer_list<float> il)	DA	Returns: An object d.	Postconditions: d.o == o and d.p == vector<float>(il).
X::surface_state_props::equal(const DA& lhs, const DA& rhs)	bool	Returns: lhs.o == rhs.o && lhs.p == rhs.p.	

Table 13 — X::surface_state_props requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::surface_state_props::not_equal(const DA& lhs, const DA& rhs)	bool	Returns: lhs.o != rhs.o lhs.p != rhs.p.	

9.2.7 surfaces requirements

[io2d.graphsurf.reqs-surfaces]

- 1 Let X be a GraphicsSurfaces type.
- 2 Let G be a GraphicsMath type.
- 3 Let IM be an object of *unspecified* type that contains visual data.
- 4 Let OU be an object of *unspecified* type that provides all functionality needed to display visual data on an output device and to process all operations required to create, maintain, and destroy the mechanism used to display visual data. [*Example*: In a windowing environment the mechanism would typically be a window. — *end example*]
- 5 Let UN be an object of *unspecified* type that provides all functionality needed to display visual data on an output device which does not process the operations required to create, maintain, and destroy the mechanism used to display visual data. [*Note*: This type lets the user draw on an existing output mechanism which the user manages. — *end note*]
- 6 The types OU and UN may be the same type.
- 7 The definition of an output device is provided in 16.3.9.
- 8 Table 16 describes the observable effects of the member functions of X::surfaces.
- 9 Table 14 defines the required *typedef-names* in X::surfaces, which are identifiers for class types capable of storing all data required to support the corresponding class template.

Table 14 — X::surfaces typedef-names

<i>typedef-name</i>	Class data
image_surface_data_type	basic_image_surface
output_surface_data_type	basic_output_surface
unmanaged_output_surface_data_type	basic_unmanaged_output_surface

- 10 In Table 15 and Table 16, I denotes the type image_surface_data_type, O denotes the type output_surface_data_type, U denotes the type unmanaged_output_surface_data_type, BB denotes the type basic_bounding_box<G>, BP denotes the type basic_brush_props<X>, BR denotes the type basic_brush<X>, CP denotes the type basic_clip_props<X>, D denotes the type basic_dashes<X>, DP denotes the type basic_display_point<G>, FI denotes the type basic_figure_items<X>::figure_item, IMS denotes the type basic_image_surface<X>, IP denotes the type basic_interpreted_path<X>, M denotes the type basic_matrix_2d<G>, MP denotes the type basic_mask_properties<X>, OUS denotes the type basic_output_surface<X>, P denotes the type basic_point_2d<G>, RP denotes the type basic_render_props<X>, SP denotes the type basic_stroke_props<X>, and UOS denotes the type basic_unmanaged_output_surface<X>.
- 11 In order to describe the observable effects of functions contained in Table 16, Table 15 describes the types contained in X as-if they possessed certain member data.

Table 15 — X::surfaces type member data

Type	Member data	Member data type
I	im	IM
I	fmt	format
I	dm	DP
O	ou	OU

Type	Member data	Member data type
O	fmt	format
O	dm	DP
O	bb	I
O	lb	optional
O	lbp	BP
O	sc	scaling
O	ac	bool
O	rr	bool
O	rs	refresh_style
O	dfr	float
O	dc	function<void(OUS&)>
O	scc	function<void(OUS&)>
U	un	UN
U	dm	DP
U	bb	I
U	lb	optional
U	lbp	BP
U	sc	scaling
U	ac	bool
U	dc	function<void(OUS&)>
U	scc	function<void(OUS&)>

- ¹² [*Note:* In the same way that `stdin`, `stdout`, and `stderr` do not specify how they meet certain requirements, the requirements set forth in Table 16 also do not specify how they meet certain requirements, most or all of which relate to the output device. — *end note*]
- ¹³ [*Note:* Operations on objects of types IM, OU, and UO follow the C++ requirements regarding observable behavior (See: C++ 2017[intro.execution]). Successive operations on such objects are not observable unless and until the visual data of such objects can be observed, such as when the visual data is displayed on an output device or is written out to a file. As such, implementations that use graphics acceleration hardware can use batching and other deferred processing techniques to improve performance. — *end note*]

Table 16 — Graphics surfaces requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::surfaces::max_image_dimensions()</code>	DP	<i>Returns:</i> An object <code>dp</code> where <code>dp.x()</code> is the maximum width in pixels of the visual data of an object of type IM and <code>dp.y()</code> is the maximum height in pixels of the visual data of an object of type IM.	
<code>X::surfaces::max_output_dimensions()</code>	DP	<i>Returns:</i> An object <code>dp</code> where <code>dp.x()</code> is the maximum width in pixels of the visual data of an object of type OU and <code>dp.y()</code> is the maximum height in pixels of the visual data of an object of type OU.	
<code>X::surfaces::max_unmanaged_output_dimensions()</code>	DP	<i>Returns:</i> An object <code>dp</code> where <code>dp.x()</code> is the maximum width in pixels of the visual data of an object of type UN and <code>dp.y()</code> is the maximum height in pixels of the visual data of an object of type UN.	

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::surfaces::create_ image_ surface(format fmt, int w, int h)</code>		<i>Returns:</i> An object <code>i</code> .	<i>Requires:</i> <code>fmt != format::invalid, w > 0, w <= max_image_dimensions().x(), h > 0, and h <= max_image_dimensions().y()</code> . <i>Postconditions:</i> The bounds of <code>i.im</code> are <code>[0, w)</code> along the x axis and <code>[0, h)</code> along the y axis, the visual data format of <code>i.im</code> is <code>fmt</code> , <code>i.fmt == fmt</code> , and <code>i.dm == DP(x, y)</code> . The values of the visual data of <code>i.im</code> are <i>unspecified</i> .

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<pre>X::surfaces::create_image_surface(filesystem::path f, image_file_format iff, format fmt) X::surfaces::create_image_surface(filesystem::path f, image_file_format iff, format fmt, error_code& ec) noexcept</pre>		<i>Returns:</i> An object <i>i</i> .	<p><i>Requires:</i> <i>f</i> is a file, the contents of <i>f</i> are valid data in the data format (16.2.1) specified by <i>iff</i>, the bounds of the visual data contained in the contents of <i>f</i> do not exceed the values returned by <code>max_image_dimensions()</code>, and <code>fmt != format::invalid</code>.</p> <p><i>Postconditions:</i> The visual data format of <i>i.im</i> is <i>fmt</i>. The bounds and visual data of <i>i.im</i> are the result of processing the contents of <i>f</i> and transforming the visual data it contains into the visual data format <i>fmt</i>. <code>i.fmt == fmt</code>. <i>i.dm</i> is equal to the result of creating an object of type DP using the bounds obtained from processing the contents of <i>f</i>.</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
(Continued...)			<p><i>Throws:</i> As specified in Error reporting (Clause 4).</p> <p><i>Error conditions:</i> Any error that could result from trying to access <code>f</code>, open <code>f</code> for reading, or reading data from <code>f</code>. Other errors, if any, are implementation-defined.</p>
<pre>X::surfaces::save(I& i, filesystem::path f, image_file_format iff) X::surfaces::save(I& i, filesystem::path f, image_file_format iff, error_code& ec) noexcept</pre>	void	Any pending rendering and composing operations (16.3.2) on <code>i.im</code> are performed. The visual data of <code>i.im</code> is written to <code>f</code> in the data format specified by <code>iff</code> .	<p><i>Requires:</i> <code>f</code> shall be a valid path to a file. It is not required that the file exist provided that the other components of the path are valid.</p> <p><i>Throws:</i> As specified in Error reporting (Clause 4).</p> <p><i>Error conditions:</i> Any error that could result from trying to access <code>f</code>, open <code>f</code> for writing, or write data to <code>f</code>. Other errors, if any, are implementation-defined.</p>
<pre>X::surfaces::format(const I& i) noexcept</pre>	std::format	<i>Returns:</i> <code>i.fmt</code> .	
<pre>X::surfaces::dimension(const I& i) noexcept</pre>	std::dm	<i>Returns:</i> <code>i.dm</code> .	

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>X::surfaces::clear(I& void i)</code>		<i>Effects:</i> Equivalent to: <code>X::surfaces::paint(i, BR(rgba_-, color::transparent_black), nullopt, make_optional<RP>(antialias::none, M(), compositing_op::clear);</code>	
<code>X::surfaces::paint(I& void i, const BB& b, const BP& bp, const RP& rp, const CP& cl);</code>		<i>Effects:</i> Perform the painting operation on <code>i.im</code> as specified in 16.3.4. <code>b</code> is the source brush. <code>bp</code> is the brush properties. <code>rp</code> is the surface properties. <code>cl</code> is the clip properties.	
<code>X::surfaces::stroke(I&void i, const BB& b, const IP& ip, const BP& bp, const SP& sp, const D& d, const RP& rp, const CP& cl);</code>		<i>Effects:</i> Perform the stroking operation on <code>i.im</code> as specified in 16.3.6.	
<code>X::surfaces::fill(I& void i, const BB& b, const IP& ip, const BP& bp, const RP& rp, const CP& cl);</code>		<i>Effects:</i> Perform the filling operation on <code>i.im</code> as specified in 16.3.5.	
<code>X::surfaces::mask(I& void i, const BB& b, const BB& m, const BP& bp, const MP& mp, const RP& rp, const CP& cl);</code>		<i>Effects:</i> Perform the masking operation on <code>i.im</code> as specified in 16.3.7.	

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<pre>X::surfaces::create_0 output_surface(int pw, int ph, io2d::format pfmt, io2d::scaling scl, io2d::refresh_style rs, float fps) X::surfaces::create_1 output_surface(int pw, int ph, io2d::format pfmt, error_code& ec, io2d::scaling scl, io2d::refresh_style rs, float fps) noexcept</pre>		<i>Returns:</i> An object <i>o</i> .	<p><i>Requires:</i> $pw > 0$, $pw \leq \min(\max_image_dimensions().x(), \max_output_dimensions().x())$, $ph > 0$, $ph \leq \min(\max_image_dimensions().y(), \max_output_dimensions().y())$, $pfmt \neq \text{format}::\text{invalid}$, and $fps < 0.0f$.</p> <p><i>Postconditions:</i> The bounds of <i>o.ou</i> are $[0, pw)$ along the <i>x</i> axis and $[0, ph)$ along the <i>y</i> axis. The visual data format of <i>o.ou</i> is <i>fmt</i> or, if <i>pfmt</i> is not supported for <i>o.ou</i> then an implementation-defined visual data format. <i>o.fmt</i> is set to the <i>format</i> enumerator that corresponds to the visual data format of <i>o.ou</i>, which may be a value in <i>X::additional_formats</i>. $o.dm == DP(pw, ph)$. The values of the visual data of <i>o.ou</i> are <i>unspecified</i>.</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
(Continued...)			<p>The bounds of <code>o.bb.im</code> are <code>[0, pw)</code> along the x axis and <code>[0, ph)</code> along the y axis, the visual data format of <code>o.bb.im</code> is <code>pfmt</code>, <code>o.bb.fmt == pfmt</code>, and <code>o.bb.dm == DP(pw, ph)</code>. The values of the visual data of <code>o.bb.im</code> are <i>unspecified</i>. <code>o.lb.value() == BR(rgba_color::black)</code>. <code>o.lbp == BP()</code>. <code>o.sc == scaling::letterbox</code>. <code>o.ac == false</code>. <code>o.rr == false</code>. <code>o.rs == rs</code>. <code>o.dfr == fps</code>. <code>o.dc == nullptr</code>. <code>o.scc == nullptr</code>. <i>Remarks:</i> Implementations may defer the creation of <code>o.ou</code> and <code>o.bb</code> until <code>begin_show(o, ...)</code> is called. Implementations may defer the creation of the visual data of the object contained in <code>o.lb</code> until it is used.</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
(Continued...)			<p>When <code>o.ou</code> and <code>o.bb</code> are created, the implementation shall ensure that the values of <code>o.dm</code> and <code>o.bb.dm</code> are set to the bounds of <code>o.ou</code>, and if either value changed it shall then invoke <code>o.scc</code> if <code>o.scc != nullptr</code>.</p> <p><i>Throws:</i> As specified in Error reporting (Clause 4).</p> <p><i>Error conditions:</i> <code>errc::not_supported</code> if creating <code>o</code> would exceed the maximum number of simultaneous <code>basic_output_surface</code> objects or combination of <code>basic_output_surface</code> objects and <code>basic_unmanaged_output_surface</code> objects supported by the implementation (See: 16.3.9).</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<pre>X::surfaces::create_0 output_surface(int pw, int ph, io2d::format pfmt, int pdw, int pdh, io2d::format pdfmt, io2d::scaling scl, io2d::refresh_style rs, float fps) X::surfaces::create_1 output_surface(int pw, int ph, io2d::format pfmt, int pdw, int pdh, io2d::format pdfmt, error_code& ec, io2d::scaling scl, io2d::refresh_style rs, float fps) noexcept</pre>		<p><i>Returns:</i> An object <code>o</code>.</p>	<p><i>Requires:</i> <code>pw > 0</code>, <code>pw <= max_image_dimensions().x()</code>, <code>ph > 0</code>, <code>ph <= max_image_dimensions().y()</code>, <code>pfmt != format::invalid</code>, <code>pdw > 0</code>, <code>pdh <= max_output_dimensions().x()</code>, <code>pdh > 0</code>, <code>pdh <= max_output_dimensions().y()</code>, <code>pdfmt != format::invalid</code>, and <code>fps < 0.0f</code>.</p> <p><i>Postconditions:</i> The bounds of <code>o.ou</code> are <code>[0, pdw)</code> along the <i>x</i> axis and <code>[0, pdh)</code> along the <i>y</i> axis. The visual data format of <code>o.ou</code> is <code>pdfmt</code> or, if <code>pdfmt</code> is not supported for <code>o.ou</code> then an implementation-defined visual data format. <code>o.fmt</code> is set to the <code>format</code> enumerator that corresponds to the visual data format of <code>o.ou</code>, which may be a value in <code>X::additional_formats</code>. <code>o.dm == DP(pdw, pdh)</code>. The values of the visual data of <code>o.ou</code> are <i>unspecified</i>.</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
(Continued...)			<p>The bounds of <code>o.bb.im</code> are <code>[0, pw)</code> along the x axis and <code>[0, ph)</code> along the y axis, the visual data format of <code>o.bb.im</code> is <code>pfmt</code>, <code>o.bb.fmt == pfmt</code>, and <code>o.bb.dm == DP(pw, ph)</code>. The values of the visual data of <code>o.bb.im</code> are <i>unspecified</i>. <code>o.lb.value() == BR(rgba_color::black)</code>. <code>o.lbp == BP()</code>. <code>o.sc == scaling::letterbox</code>. <code>o.ac == false</code>. <code>o.rr == false</code>. <code>o.rs == rs</code>. <code>o.dfr == fps</code>. <code>o.dc == nullptr</code>. <code>o.scc == nullptr</code>. <i>Remarks:</i> Implementations may defer the creation of <code>o.ou</code> and <code>o.bb</code> until <code>begin_show(o, ...)</code> is called. Implementations may defer the creation of the visual data of the object contained in <code>o.lb</code> until it is used.</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
(Continued...)			<p>When <code>o.ou</code> and <code>o.bb</code> are created, the implementation shall ensure that the values of <code>o.dm</code> and <code>o.bb.dm</code> are set to the bounds of <code>o.ou</code>, and if either value changed it shall then invoke <code>o.scc</code> if <code>o.scc != nullptr</code>.</p> <p><i>Throws:</i> As specified in Error reporting (Clause 4).</p> <p><i>Error conditions:</i> <code>errc::not_supported</code> if creating <code>o</code> would exceed the maximum number of simultaneous <code>basic_output_surface</code> objects or combination of <code>basic_output_surface</code> objects and <code>basic_unmanaged_output_surface</code> objects supported by the implementation (See: 16.3.9).</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<pre>X::surfaces::begin_ show(O& o, OUS& sfc) X::surfaces::begin_ show(O& o, OUS& src, error_code& ec) noexcept</pre>	void	<p>Performs the following actions in a continuous loop:</p> <ol style="list-style-type: none"> 1. Handle any implementation and host environment matters, including updating the value of <code>o.dm</code> if the output device bounds have changed; then, 2. If the value of <code>o.dm</code> changed and <code>o.scc != nullptr</code>, invoke <code>o.scc</code>; then, 3. If <code>o.rr == true</code> or the values of <code>o.rs</code> and <code>o.dfr</code> require that <code>o.dc</code> be called: <ol style="list-style-type: none"> a) Set <code>o.rr</code> to <code>false</code>; then, b) If <code>o.ac == true</code>, invoke <code>clear(o.bb)</code>; then, c) Invoke <code>o.dc</code>; then, d) Transfer <code>o.bb.im</code> to <code>o.ou</code>, performing the scaling and letterboxing, if any, required by <code>o.sc</code> and the color space conversion, if any, required to transform <code>o.bb.im</code> from <code>o.bb.fmt</code> to <code>o.fmt</code>. 	<p><i>Requires:</i> <code>sfc.data() == o</code>.</p> <p><i>Throws:</i> As specified in Error reporting (Clause 4).</p> <p><i>Error conditions:</i> <code>errc::not_supported</code> if creating or displaying <code>o.im</code> would exceed the maximum number of simultaneous <code>basic_output_surface</code> objects or combination of <code>basic_output_surface</code> objects and <code>basic_unmanaged_output_surface</code> objects supported by the implementation (See: 16.3.9).</p>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
(Continued...)			<i>Remarks:</i> All observable effects to visual data produced as a result of steps 3-b and 3-c above are as-if they were applied to <code>o.bb.im</code> in those steps. Implementations may apply those observable effects directly to <code>o.ou</code> provided that they do so as-if the scaling, letterboxing, and color space conversion behavior specified in 3-d occurs. [<i>Note:</i> This allows implementations which do not wish to use a back buffer the freedom to do so. — <i>end note</i>]
<code>X::surfaces::end_show(O&o)</code>	<code>void</code>	Initiates the process of exiting the continuous loop resulting from the invocation of <code>begin_show(o, ...)</code> . Implementations should follow any procedures that the host environment requires in order to stop the continuous loop without error. If the continuous loop resulting from the invocation of <code>begin_show(o, ...)</code> is not executing or is already exiting due to a previous call to this function, this function does nothing.	<i>Remarks:</i> This function shall not wait until the continuous loop from <code>begin_show(o, ...)</code> ends before returning. [<i>Note:</i> The correct way to exit the <code>begin_show(o, ...)</code> continuous loop is to call this function from <code>o.dc</code> or from another thread. — <i>end note</i>]

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>dimensions(0& o, const DP& dm)</code>	void	Attempts to change the bounds of <code>o.bb.im</code> to <code>dm</code> . If successful, <code>o.bb.dm == dm</code> , otherwise there are no effects.	<i>Requires:</i> <code>dm.x() > 0,</code> <code>dm.x() <= max_image_dimensions().x(),</code> <code>dm.y() > 0,</code> <code>dm.y() <= max_image_dimensions().y()</code>
<code>output_dimensions(0& o, const DP& dm)</code>	void	Attempts to change the bounds of <code>o.ou</code> to <code>dm</code> . If successful sets <code>o.dm</code> to the value of <code>dm</code> and then invokes <code>o.scc</code> unless <code>o.scc != nullptr</code> , otherwise there are no effects.	<i>Requires:</i> <code>dm.x() > 0,</code> <code>dm.x() <= max_output_dimensions().x(),</code> <code>dm.y() > 0,</code> <code>dm.y() <= max_output_dimensions().y()</code>
<code>scaling(0& o, io2d::scaling sc)</code>	void		<i>Postconditions:</i> <code>o.sc == sc.</code>
<code>refresh_style(0& o, io2d::refresh_style rs)</code>	void		<i>Postconditions:</i> <code>o.rs == rs.</code>
<code>desired_frame_rate(0& o, float dfr)</code>	void		<i>Requires:</i> <code>dfr > 0.0f.</code> <i>Postconditions:</i> <code>o.dfr == dfr.</code>
<code>letterbox_brush(0& o, const optional<BB>& lb, const optional<BP>& lbp)</code>	void		<i>Postconditions:</i> If <code>lb.has_value() == true</code> then <code>o.lb == lb.value()</code> , otherwise <code>o.lb == BB(rgba_color::black)</code> . If <code>lbp.has_value() == true</code> then <code>o.lpb == lbp.value()</code> , otherwise <code>o.lpb == BP()</code> .
<code>letterbox_brush_properties(0& o, const optional<BP>& lbp)</code>	void		<i>Postconditions:</i> If <code>lbp.has_value() == true</code> then <code>o.lpb == lbp.value()</code> , otherwise <code>o.lpb == BP()</code> .
<code>auto_clear(0& o, bool ac)</code>	void		<i>Postconditions:</i> <code>o.ac == ac.</code>
<code>redraw_required(0& o, bool rr)</code>	void		<i>Postconditions:</i> <code>o.rr == rr.</code>

Table 16 — Graphics surfaces requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
<code>draw_callback(O& o, function<void(OUS&)> dc)</code>	void		<i>Postconditions:</i> o.dc == dc.
<code>size_change_callback(O& o, function<void(OUS&)> scc)</code>	void		<i>Postconditions:</i> o.scc == scc.
<code>X::surfaces::clear(O& void o)</code>	void	<i>Effects:</i> Equivalent to: <code>paint(o.bb)</code> .	
<code>X::surfaces::paint(O& void o, const BB& b, const BP& bp, const RP& rp, const CP& cl);</code>	void	Perform the painting operation on <code>i.im</code> as specified in 16.3.4.	
<code>X::surfaces::stroke(I&void i, const BB& b, const IP& ip, const BP& bp, const SP& sp, const D& d, const RP& rp, const CP& cl);</code>	void	Perform the stroking operation on <code>i.im</code> as specified in 16.3.6.	
<code>X::surfaces::fill(I& void i, const BB& b, const IP& ip, const BP& bp, const RP& rp, const CP& cl);</code>	void	Perform the filling operation on <code>i.im</code> as specified in 16.3.5.	
<code>X::surfaces::mask(I& void i, const BB& b, const BB& m, const BP& bp, const MP& mp, const RP& rp, const CP& cl);</code>	void	Perform the masking operation on <code>i.im</code> as specified in 16.3.5.	
<code>X::surfaces::dimension(O& o) noexcept</code>	const BP	<i>Returns:</i> o.bb.dm.	
<code>X::surfaces::output_dimensions(const O& o) noexcept</code>	DP	<i>Returns:</i> o.dm.	
<code>X::surfaces::refresh_io2d::refresh_style(const O& o) noexcept</code>	DP	<i>Returns:</i> o.rs	
<code>X::surfaces::desired_frame_rate(const O& o) noexcept</code>	float	<i>Returns:</i> o.dfr.	
<code>X::surfaces::scaling(const O& o) noexcept</code>	const BP	<i>Returns:</i> o.sc.	
<code>X::surfaces::letterbox_optional<BB> brush(const O& o) noexcept</code>	BB	<i>Returns:</i> o.lb.	
<code>X::surfaces::letterbox_props(const O& o) noexcept</code>	BP	<i>Returns:</i> o.lbp.	
<code>X::surfaces::auto_clear(const O& o) noexcept</code>	bool	<i>Returns:</i> o.ac.	

Type	Member data	Member data type
font_data_type	fs	font_style
font_data_type	merging	bool
text_props_data_type	scale	float
text_props_data_type	fsu	font_size_units
text_props_data_type	fontsize	float
text_props_data_type	kern	bool
text_props_data_type	aa	font_antialias
text_props_data_type	stretch	font_stretching
text_props_data_type	strike	bool
text_props_data_type	fl	font_line

Table 19 — X::text requirements

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::text::create_font(string name, font_size_units fsu, float size, generic_font_names gfn, io2d::font_weight fw, font_style fs, bool merging)	FD	Returns: An object fd.	<p><i>Postconditions:</i></p> <p>fd.fsu == fsu, fd.fontsize == size, fd.fw == fw, fd.fs == fs, and fd.merging == merging. The value of fd.family is name when an OFF Font with that family name is present, in which case the value of fd.font is that OFF Font. Otherwise the value of fd.family is the family name of the OFF Font selected by the implementation in an unspecified manner and the value of fd.font is that OFF Font. In the latter case, implementations should use the value of gfn together with the values of the other arguments to select an OFF Font.</p>

Table 19 — X::text requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::text::create_font(P file, string name, font_size_units fsu, float size, io2d::font_weight fw, font_style fs, bool merging)	FD	Returns: An object fd.	<p><i>Postconditions:</i> font is the OFF Font contained in file, fd.fsu == fsu, fd.fontsize == size, fd.fw == fw, fd.fs == fs, and fd.merging == merging. Where file is not an OFF Font Font Collection, the value of fd.family is the family name of the OFF Font contained in file. Where file is an OFF Font Font Collection, the value of fd.family is name if a font with that family name is contained in file, otherwise it is the family name of the font contained in that file selected by the implementation in an unspecified manner. In the latter case implementations should select the font that most closely corresponds to the values of the other arguments.</p>

Table 19 — X::text requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::text::create_font(generic_font_names gfn, font_size_units fsu, float size, font_weight fw, font_style fs)	FD	Returns: An object fd.	<i>Postconditions:</i> fd.fsu == fsu, fd.fontsize == size, and fd.merging == merging. The value of fd.family is the string that is the name of the enumerator value gfn. The value of fd.font shall be determined as described in 12.9.1, taking into account the values of fw and fs. If value chosen for fd.font is such that the value for fd.fw would better match an enumerator different than that specified by fw the value of fd.fw is the better match, otherwise it is fw. If value chosen for fd.font is such that the value for fd.fs would better match an enumerator different than that specified by fs the value of fd.fs is the better match, otherwise it is fs.
X::text::font_size(FD& data, font_size_units fsu, float size)	void		<i>Postconditions:</i> data.fsu == fsu and data.fontsize == size.
X::text::merging(FD& data, bool m)	void		<i>Postconditions:</i> data.merging == m.

Table 19 — X::text requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::text::font_size(const FD& data)	float	Returns: data.fontsize.	
X::text::family(const FD& data)	string	Returns: data.family.	
X::text::size_units(const FD& data)	font_size_units	Returns: data.fsu.	
X::text::weight(const FD& data)	font_weight	Returns: data.fw.	
X::text::style(const FD& data)	font_style	Returns: data.fs.	
X::text::merging(const FD& data)	bool	Returns: data.merging.	
X::text::create_text_props(float scl, font_size_units fsu, float size, bool kern, font_antialias aa, font_stretching stretch, bool strike_through, font_line fl)	TP	Returns: An object tp.	Postconditions: tp.scale == scl, tp.fsu == fsu, tp.fontsize == size, tp.kern == kern, tp.aa == aa, tp.stretch == stretch, tp.strike == strike_through, and tp.fl == fl.
X::text::scale(TP& data, float scl)	void		Postconditions: data.scale == scl.
X::text::font_size(TP& data, font_size_units fsu, float sz)	void		Postconditions: data.fsu == fsu and data.fontsize == sz.
X::text::kerning(TP& data, bool k)	void		Postconditions: data.kern == k.
X::text::antialiasing(TP& data, font_antialias aa)	void		Postconditions: data.aa == aa.
X::text::stretching(TP& data, font_stretching fs)	void		Postconditions: data.stretch == fs.
X::text::strike_through(TP& data, bool st)	void		Postconditions: data.strike == st.
X::text::line(TP& data, font_line fl)	void		Postconditions: data.fl == fl.
X::text::scale(const TP& data)	float	Returns: data.scale.	

Table 19 — X::text requirements (continued)

Expression	Return type	Operational semantics	Assertion/note pre-/post-condition
X::text::font_size(const TP& data)	float	<i>Returns:</i> data.fontsize.	
X::text::size_units(const TP& data)	font_size_units	<i>Returns:</i> data.fsu.	
X::text::kerning(const TP& data)	bool	<i>Returns:</i> data.kern.	
X::text::antialiasing(const TP& data)	font_antialias	<i>Returns:</i> data.aa.	
X::text::stretching(const TP& data)	font_stretching	<i>Returns:</i> data.stretch.	
X::text::stroke_through(const TP& data)	bool	<i>Returns:</i> data.strike.	
X::text::line(const TP& data)	font_line	<i>Returns:</i> data.fl.	
X::text::create_font_database()	FDB	<i>Returns:</i> An object fdb.	
X::text::get_families(const FDB& data)	vector<string>	<i>Returns:</i> A vector<string> containing the family names of all font families available in the host environment.	

10 Linear algebra [io2d.linearalgebra]

10.1 Class `basic_point_2d` [io2d.point2d]

10.1.1 `basic_point_2d` description [io2d.point2d.intro]

- ¹ The class template `basic_point_2d` is used as both a point and as a two-dimensional Euclidean vector.
- ² It has an *x coordinate* of type `float` and a *y coordinate* of type `float`.
- ³ The data are stored in an object of type `typename GraphicsMath::point_2d_data_type`. It is accessible using the data member functions.

10.1.2 `basic_point_2d` synopsis [io2d.point2d.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsMath>
    class basic_point_2d {
    public:
        using data_type = typename GraphicsMath::point_2d_data_type;

        // 10.1.3, constructors:
        basic_point_2d() noexcept;
        basic_point_2d(float x, float y) noexcept;
        basic_point_2d(const typename GraphicsMath::point_2d_data_type& data) noexcept;

        // 10.1.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 10.1.5, modifiers:
        void x(float val) noexcept;
        void y(float val) noexcept;

        // 10.1.6, observers:
        float x() const noexcept;
        float y() const noexcept;
        float dot(const basic_point_2d& other) const noexcept;
        float magnitude() const noexcept;
        float magnitude_squared() const noexcept;
        float angular_direction() const noexcept;
        basic_point_2d to_unit() const noexcept;

        // 10.1.7, member operators:
        basic_point_2d& operator+=(const basic_point_2d& rhs) noexcept;
        basic_point_2d& operator+=(float rhs) noexcept;
        basic_point_2d& operator-=(const basic_point_2d& rhs) noexcept;
        basic_point_2d& operator-=(float rhs) noexcept;
        basic_point_2d& operator*=(const basic_point_2d& rhs) noexcept;
        basic_point_2d& operator*=(float rhs) noexcept;
        basic_point_2d& operator/=(const basic_point_2d& rhs) noexcept;
        basic_point_2d& operator/=(float rhs) noexcept;
    };

    // 10.1.8, non-member operators:
    template <class GraphicsMath>
    bool operator==(const basic_point_2d<GraphicsMath>& lhs,
        const basic_point_2d<GraphicsMath>& rhs) noexcept;
    template <class GraphicsMath>
    bool operator!=(const basic_point_2d<GraphicsMath>& lhs,
        const basic_point_2d<GraphicsMath>& rhs) noexcept;
}
```

```

template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator+(
    const basic_point_2d<GraphicsMath>& val) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator+(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator-(
    const basic_point_2d<GraphicsMath>& val) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator-(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(
    const basic_point_2d<GraphicsMath>& lhs,
    float rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(float lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator/(
    const basic_point_2d<GraphicsMath>& lhs,
    float rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator/(float lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator/(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_point_2d<GraphicsMath>& rhs) noexcept;
}

```

10.1.3 basic_point_2d constructors

[io2d.point2d.cons]

```
basic_point_2d() noexcept;
```

1 *Effects:* Constructs an object of type `basic_point_2d`.

2 *Postconditions:* `data() == GraphicsMath::create_point_2d()`.

```
basic_point_2d(float x, float y) noexcept;
```

3 *Effects:* Constructs an object of type `basic_point_2d`.

4 *Postconditions:* `data() == GraphicsMath::create_point_2d(x, y)`.

```
basic_point_2d(const data_type& d) noexcept;
```

5 *Effects:* Constructs an object of type `basic_point_2d`.

6 *Postconditions:* `data() == d`.

10.1.4 basic_point_2d accessors

[io2d.point2d.accessors]

```
const data_type& data() const noexcept;
```

```
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_point_2d` object's data object (See: 10.1.1).

10.1.5 basic_point_2d modifiers**[io2d.point2d.modifiers]**

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

```
1   Effects: Equivalent to GraphicsMath::x(data(), val);
```

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

```
2   Effects: Equivalent to GraphicsMath::y(data(), val);
```

10.1.6 basic_point_2d observers**[io2d.point2d.observers]**

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

```
1   Returns: GraphicsMath::x(data()).
```

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

```
2   Returns: GraphicsMath::y(data()).
```

```
float dot(const basic_point_2d& other) const noexcept;
```

```
3   Returns: GraphicsMath::dot(data(), other).
```

```
float magnitude() const noexcept;
```

```
4   Returns: GraphicsMath::magnitude(data()).
```

```
float magnitude_squared() const noexcept;
```

```
5   Returns: GraphicsMath::magnitude_squared(data()).
```

```
float angular_direction() const noexcept;
```

```
6   Returns: GraphicsMath::angular_direction(data()).
```

```
basic_point_2d to_unit() const noexcept;
```

```
7   Returns: basic_point_2d(GraphicsMath::to_unit(data())).
```

10.1.7 basic_point_2d member operators**[io2d.point2d.member.ops]**

```
basic_point_2d& operator+=(const basic_point_2d& rhs) noexcept;
```

```
1   Effects: Equivalent to data() = GraphicsMath::add(data(), rhs.data());
```

```
2   Returns: *this.
```

```
basic_point_2d& operator-=(const basic_point_2d& rhs) noexcept;
```

```
3   Effects: Equivalent to data() = GraphicsMath::subtract(data(), rhs.data());
```

```
4   Returns: *this.
```

```
basic_point_2d& operator*=(float rhs) noexcept;
```

```
5   Effects: Equivalent to data() = GraphicsMath::multiply(data(), rhs);
```

```
6   Returns: *this.
```

```
basic_point_2d& operator*=(const basic_point_2d& rhs) noexcept;
```

```
7   Effects: Equivalent to data() = GraphicsMath::multiply(data(), rhs.data());
```

```
8   Returns: *this.
```

```
basic_point_2d& operator/=(float rhs) noexcept;
```

```
9   Effects: Equivalent to: data() = GraphicsMath::divide(data(), rhs);
```

```
10  Returns: *this.
```

```
basic_point_2d& operator/=(const basic_point_2d& rhs) noexcept;
```

```
11  Effects: Equivalent to: data() = GraphicsMath::divide(data(), rhs.data());
```

```
12  Returns: *this.
```

10.1.8 basic_point_2d non-member operators**[io2d.point2d.ops]**

```

bool operator==(const basic_point_2d& lhs, const basic_point_2d& rhs) noexcept;
1   Returns: GraphicsMath::equal(lhs.data(), rhs.data()).

bool operator!=(const basic_point_2d& lhs, const basic_point_2d& rhs) noexcept;
2   Returns: GraphicsMath::not_equal(lhs.data(), rhs.data()).

basic_point_2d operator+(const basic_point_2d& val) noexcept;
3   Returns: val.

basic_point_2d operator+(const basic_point_2d& lhs, const basic_point_2d& rhs) noexcept;
4   Returns: basic_point_2d(GraphicsMath::add(lhs.data(), rhs.data())).

basic_point_2d operator-(const basic_point_2d& val) noexcept;
5   Returns: basic_point_2d(GraphicsMath::negate(val.data())).

basic_point_2d operator-(const basic_point_2d& lhs, const basic_point_2d& rhs) noexcept;
6   Returns: basic_point_2d(GraphicsMath::subtract(lhs.data(), rhs.data())).

basic_point_2d operator*(const basic_point_2d& lhs, const basic_point_2d& rhs) noexcept;
7   Returns: basic_point_2d(GraphicsMath::multiply(lhs.data(), rhs.data())).

basic_point_2d operator*(const basic_point_2d& lhs, float rhs) noexcept;
8   Returns: basic_point_2d(GraphicsMath::multiply(lhs.data(), rhs)).

basic_point_2d operator*(float lhs, const basic_point_2d& rhs) noexcept;
9   Returns: basic_point_2d(GraphicsMath::multiply(lhs, rhs.data())).

basic_point_2d operator/(const basic_point_2d& lhs, const basic_point_2d& rhs) noexcept;
10  Requires: rhs.x() != 0.0f and rhs.y() != 0.0f.
11  Returns: basic_point_2d(GraphicsMath::divide(lhs.data(), rhs.data())).

basic_point_2d operator/(const basic_point_2d& lhs, float rhs) noexcept;
12  Requires: rhs != 0.0f.
13  Returns: basic_point_2d(GraphicsMath::divide(lhs.data(), rhs)).

basic_point_2d operator/(float lhs, const basic_point_2d& rhs) noexcept;
14  Requires: rhs.x() != 0.0f and rhs.y() != 0.0f.
15  Returns: basic_point_2d(GraphicsMath::divide(lhs, rhs.data())).

```

10.2 Class basic_matrix_2d**[io2d.matrix2d]****10.2.1 basic_matrix_2d description****[io2d.matrix2d.intro]**

- 1 The class template `basic_matrix_2d` represents a three row by three column matrix. Its purpose is to perform affine transformations.
- 2 The matrix is composed of nine `float` values: `m00`, `m01`, `m02`, `m10`, `m11`, `m12`, `m20`, `m21`, and `m22`. The ordering of these `float` values in the `data` of a `basic_matrix_2d` object is unspecified.
- 3 The specification of the `basic_matrix_2d` class, as described in this subclause, uses the following ordering:

```

[ [ m00 m01 m02 ] ]
[ [ m10 m11 m12 ] ]
[ [ m20 m21 m22 ] ]

```
- 4 [*Note:* The naming convention and the layout shown above are consistent with a row-major layout. Though the naming convention is fixed, the unspecified layout allows for a column-major layout (or any other layout, though row-major and column-major are the only layouts typically used). — *end note*]
- 5 The performance of any mathematical operation upon a `basic_matrix_2d` shall be carried out as-if the omitted third column data members were present with the values prescribed in the previous paragraph.

- ⁶ The data are stored in an object of type `typename GraphicsMath::matrix_2d_data_type`. It is accessible using the data member functions.

10.2.2 basic_matrix_2d synopsis

[io2d.matrix2d.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsMath>
    class basic_matrix_2d {
    public:
        using data_type = typename GraphicsMath::matrix_2d_data_type;

        // 10.2.3, constructors:
        basic_matrix_2d() noexcept;
        basic_matrix_2d(float v00, float v01, float v10, float v11, float v20, float v21) noexcept;
        basic_matrix_2d(const typename GraphicsMath::matrix_2d_data_type& v) noexcept;

        // 10.2.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 10.2.5, static factory functions:
        static basic_matrix_2d create_translate(const basic_point_2d<GraphicsMath>& v) noexcept;
        static basic_matrix_2d create_scale(const basic_point_2d<GraphicsMath>& v) noexcept;
        static basic_matrix_2d create_rotate(float radians) noexcept;
        static basic_matrix_2d create_rotate(float radians,
            const basic_point_2d<GraphicsMath>& origin) noexcept;
        static basic_matrix_2d create_reflect(float radians) noexcept;
        static basic_matrix_2d create_shear_x(float factor) noexcept;
        static basic_matrix_2d create_shear_y(float factor) noexcept;

        // 10.2.6, modifiers:
        void m00(float v) noexcept;
        void m01(float v) noexcept;
        void m10(float v) noexcept;
        void m11(float v) noexcept;
        void m20(float v) noexcept;
        void m21(float v) noexcept;
        basic_matrix_2d& translate(const basic_point_2d<GraphicsMath>& v) noexcept;
        basic_matrix_2d& scale(const basic_point_2d<GraphicsMath>& v) noexcept;
        basic_matrix_2d& rotate(float radians) noexcept;
        basic_matrix_2d& rotate(float radians, const basic_point_2d<GraphicsMath>& origin) noexcept;
        basic_matrix_2d& reflect(float radians) noexcept;
        basic_matrix_2d& shear_x(float factor) noexcept;
        basic_matrix_2d& shear_y(float factor) noexcept;

        // 10.2.7, observers:
        float m00() const noexcept;
        float m01() const noexcept;
        float m10() const noexcept;
        float m11() const noexcept;
        float m20() const noexcept;
        float m21() const noexcept;
        bool is_finite() const noexcept;
        bool is_invertible() const noexcept;
        float determinant() const noexcept;
        basic_matrix_2d inverse() const noexcept;
        basic_point_2d<GraphicsMath> transform_pt(const basic_point_2d<GraphicsMath>& pt)
            const noexcept;

        // 10.2.8, member operators:
        basic_matrix_2d& operator*=(const basic_matrix_2d& other) noexcept;
    };
};
```

```

// 10.2.9, member operators:
template <class GraphicsMath>
basic_matrix_2d<GraphicsMath> operator*(
    const basic_matrix_2d<GraphicsMath>& lhs,
    const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
basic_point_2d<GraphicsMath> operator*(
    const basic_point_2d<GraphicsMath>& lhs,
    const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator==(const basic_matrix_2d<GraphicsMath>& lhs,
    const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator!=(const basic_matrix_2d<GraphicsMath>& lhs,
    const basic_matrix_2d<GraphicsMath>& rhs) noexcept;
}

```

10.2.3 basic_matrix_2d constructors

[io2d.matrix2d.cons]

```
basic_matrix_2d() noexcept;
```

- 1 *Effects:* Constructs an object of type `basic_matrix_2d`.
2 [*Note:* The resulting matrix is the identity matrix. — *end note*]
3 *Postconditions:* `data() == GraphicsMath::create_matrix_2d()`.

```
basic_matrix_2d(float m00, float m01, float m10, float m11,
    float m20, float m21) noexcept;
```

- 4 *Effects:* Constructs an object of type `basic_matrix_2d`.
5 *Postconditions:* `data() == GraphicsMath::create_matrix_2d(m00, m01, m10, m11, m20, m21)`.

```
basic_matrix_2d(const data_type& v) noexcept;
```

- 6 *Effects:* Constructs an object of type `basic_matrix_2d`.
7 *Postconditions:* `data() == v`.

10.2.4 basic_matrix_2d accessors

[io2d.matrix2d.accessors]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

- 1 *Returns:* A reference to the `basic_matrix_2d` object's data object (See: 10.2.1).

10.2.5 basic_matrix_2d static factory functions

[io2d.matrix2d.staticfactories]

```
static basic_matrix_2d create_translate(basic_point_2d<GraphicsMath> v) noexcept;
```

- 1 *Returns:* `basic_matrix_2d(GraphicsMath::create_translate(v.data()))`.

```
static basic_matrix_2d create_scale(basic_point_2d<GraphicsMath> v) noexcept;
```

- 2 *Returns:* `basic_matrix_2d(GraphicsMath::create_scale(v.data()))`.

```
static basic_matrix_2d create_rotate(float radians) noexcept;
```

- 3 *Returns:* `basic_matrix_2d(GraphicsMath::create_rotate(radians))`.

```
static basic_matrix_2d create_rotate(float radians, basic_point_2d<GraphicsMath> origin) noexcept;
```

- 4 *Returns:* `basic_matrix_2d(GraphicsMath::create_rotate(radians, origin.data()))`.

```
static basic_matrix_2d create_reflect(float radians) noexcept;
```

- 5 *Returns:* `basic_matrix_2d(GraphicsMath::create_reflect(radians))`

```
static basic_matrix_2d create_shear_x(float factor) noexcept;
```

- 6 *Returns:* `basic_matrix_2d(GraphicsMath::create_shear_x(factor))`.

```
static basic_matrix_2d create_shear_y(float factor) noexcept;
```

```
7     Returns: basic_matrix_2d(GraphicsMath::create_shear_y(factor)).
```

10.2.6 basic_matrix_2d modifiers

[io2d.matrix2d.modifiers]

```
void m00(float v) noexcept;
```

```
1     Effects: Equivalent to GraphicsMath::m00(data(), v);
```

```
void m01(float v) noexcept;
```

```
2     Effects: Equivalent to GraphicsMath::m01(data(), v);
```

```
void m10(float v) noexcept;
```

```
3     Effects: Equivalent to GraphicsMath::m10(data(), v);
```

```
void m11(float v) noexcept;
```

```
4     Effects: Equivalent to GraphicsMath::m11(data(), v);
```

```
void m20(float v) noexcept;
```

```
5     Effects: Equivalent to GraphicsMath::m20(data(), v);
```

```
void m21(float v) noexcept;
```

```
6     Effects: Equivalent to GraphicsMath::m21(data(), v);
```

```
basic_matrix_2d& translate(basic_point_2d<GraphicsMath> v) noexcept;
```

```
7     Effects: Equivalent to data() = GraphicsMath::translate(data(), v.data());
```

```
8     Returns: *this.
```

```
basic_matrix_2d& scale(basic_point_2d<GraphicsMath> v) noexcept;
```

```
9     Effects: Equivalent to data() = GraphicsMath::scale(data(), v.data());
```

```
10    Returns: *this.
```

```
basic_matrix_2d& rotate(float radians) noexcept;
```

```
11    Effects: Equivalent to data() = GraphicsMath::rotate(data(), radians);
```

```
12    Returns: *this.
```

```
basic_matrix_2d& rotate(float radians, basic_point_2d<GraphicsMath> origin) noexcept;
```

```
13    Effects: Equivalent to data() = GraphicsMath::rotate(data(), radians, origin.data());
```

```
14    Returns: *this.
```

```
basic_matrix_2d& reflect(float radians) noexcept;
```

```
15    Effects: Equivalent to data() = GraphicsMath::reflect(data(), radians);
```

```
16    Returns: *this.
```

```
basic_matrix_2d& shear_x(float factor) noexcept;
```

```
17    Effects: Equivalent to data() = GraphicsMath::shear_x(data(), factor);
```

```
18    Returns: *this.
```

```
basic_matrix_2d& shear_y(float factor) noexcept;
```

```
19    Effects: Equivalent to data() = GraphicsMath::shear_y(factor);
```

```
20    Returns: *this.
```

10.2.7 basic_matrix_2d observers

[io2d.matrix2d.observers]

```
float m00() const noexcept;
```

```
1     Returns: GraphicsMath::m00(data()).
```



```

float m01() const noexcept;
2     Returns: GraphicsMath::m01(data()).

float m10() const noexcept;
3     Returns: GraphicsMath::m10(data()).

float m11() const noexcept;
4     Returns: GraphicsMath::m11(data()).

float m20() const noexcept;
5     Returns: GraphicsMath::m20(data()).

float m21() const noexcept;
6     Returns: GraphicsMath::m21(data()).

bool is_finite() const noexcept;
7     Returns: GraphicsMath::is_finite(data()).

bool is_invertible() const noexcept;
8     Requires: is_finite() == true.
9     Returns: GraphicsMath::is_invertible(data()).

basic_matrix_2d inverse() const noexcept;
10    Requires: is_invertible() == true.
11    Returns: basic_matrix_2d(GraphicsMath::inverse(data())).

float determinant() const noexcept;
12    Requires: is_finite() == true.
13    Returns: GraphicsMath::determinant(data()).

basic_point_2d<GraphicsMath> transform_pt(basic_point_2d<GraphicsMath> pt) const noexcept;
14    Returns: basic_point_2d<GraphicsMath>(GraphicsMath::transform_pt(data(), pt.data())).

```

10.2.8 basic_matrix_2d member operators [io2d.matrix2d.member.ops]

```

basic_matrix_2d& operator*=(const basic_matrix_2d& rhs) noexcept;
1     Effects: Equivalent to data() = GraphicsMath::multiply(data(), rhs.data());
2     Returns: *this.

```

10.2.9 basic_matrix_2d non-member operators [io2d.matrix2d.ops]

```

basic_matrix_2d operator*(const basic_matrix_2d& lhs, const basic_matrix_2d& rhs)
noexcept;
1     Returns: basic_matrix_2d(GraphicsMath::multiply(lhs.data(), rhs.data())).

basic_point_2d<GraphicsMath> operator*(basic_point_2d<GraphicsMath> v, const basic_matrix_2d& m) noexcept;
2     Returns: Equivalent to m.transform_pt(v).

bool operator==(const basic_matrix_2d& lhs, const basic_matrix_2d& rhs) noexcept;
3     Returns: GraphicsMath::equal(lhs.data(), rhs.data()).

bool operator!=(const basic_matrix_2d& lhs, const basic_matrix_2d& rhs) noexcept;
4     Returns: Equivalent to GraphicsMath::not_equal(lhs.data(), rhs.data()).

```

11 Geometry

[io2d.geometry]

11.1 Class template `basic_display_point`

[io2d.displaypt]

11.1.1 `basic_display_point` description

[io2d.displaypt.intro]

- ¹ The class template `basic_display_point` describes an integral point used to describe certain properties of surfaces.
- ² It has an *x coordinate* of type `int` and a *y coordinate* of type `int`.
- ³ The data are stored in an object of type `typename GraphicsMath::display_point_data_type`. It is accessible using the data member functions.

11.1.2 `basic_display_point` synopsis

[io2d.displaypt.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsMath>
    class basic_display_point {
    public:
        using data_type = typename GraphicsMath::display_point_data_type;

        // 11.1.3, constructors:
        basic_display_point() noexcept;
        basic_display_point(int x, int y) noexcept;
        basic_display_point(const data_type& val) noexcept;

        // 11.1.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 11.1.5, modifiers:
        void x(int val) noexcept;
        void y(int val) noexcept;

        // 11.1.6, observers:
        int x() const noexcept;
        int y() const noexcept;
    };

    // 11.1.7, operators:
    template <class GraphicsMath>
    bool operator==(const basic_display_point<GraphicsMath>& lhs,
        const basic_display_point<GraphicsMath>& rhs) noexcept;
    template <class GraphicsMath>
    bool operator!=(const basic_display_point<GraphicsMath>& lhs,
        const basic_display_point<GraphicsMath>& rhs) noexcept;
}
```

11.1.3 `basic_display_point` constructors

[io2d.displaypt.cons]

```
basic_display_point() noexcept;
```

- ¹ *Effects:* Constructs an object of type `basic_display_point`.
- ² *Postconditions:* `data() == GraphicsMath::create_display_point()`.
- ³ *Remarks:* The x coordinate is 0 and the y coordinate is 0.

```
basic_display_point(int x, int y) noexcept;
```

- ⁴ *Effects:* Constructs an object of type `basic_display_point`.
- ⁵ *Postconditions:* `data() == GraphicsMath::create_display_point(x, y)`.
- ⁶ *Remarks:* The x coordinate is x and the y coordinate is y.

```
basic_display_point(const data_type& val) noexcept;
```

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

8 *Postconditions:* `data() == val`.

9 *Remarks:* The x coordinate is `GraphicsMath::x(val)` and the y coordinate is `GraphicsMath::y(val)`.

11.1.4 `basic_display_point` accessors [io2d.displaypt.accessors]

```
const data_type& data() const noexcept;
```

```
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_display_point` object's data object (See: 11.1.1).

11.1.5 `basic_display_point` modifiers [io2d.displaypt.modifiers]

```
void x(int v) noexcept;
```

1 *Effects:* Equivalent to `GraphicsMath::x(data(), v)`;

```
void y(int v) noexcept;
```

2 *Effects:* Equivalent to `GraphicsMath::y(data(), v)`;

11.1.6 `basic_display_point` observers [io2d.displaypt.observers]

```
int x() const noexcept;
```

1 *Returns:* `GraphicsMath::x(data())`.

```
int y() const noexcept;
```

2 *Returns:* `GraphicsMath::y(data())`.

11.1.7 `basic_display_point` operators [io2d.displaypt.ops]

```
bool operator==(const basic_display_point<GraphicsMath>& lhs,  
               const basic_display_point<GraphicsMath>& rhs) noexcept;
```

1 *Returns:* `GraphicsMath::equal(lhs.data(), rhs.data())`.

```
bool operator!=(const basic_display_point<GraphicsMath>& lhs,  
               const basic_display_point<GraphicsMath>& rhs) noexcept;
```

2 *Returns:* `GraphicsMath::not_equal(lhs.data(), rhs.data())`.

11.2 Class `basic_bounding_box` [io2d.bounding_box]

11.2.1 `basic_bounding_box` description [io2d.bounding_box.intro]

1 The class template `basic_bounding_box` describes a `bounding_box`.

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

3 The data are stored in an object of type `typename GraphicsMath::bounding_box_data_type`. It is accessible using the `data` member functions.

11.2.2 `basic_bounding_box` synopsis [io2d.bounding_box.synopsis]

```
namespace std::experimental::io2d::v1 {  
    template <class GraphicsMath>  
    class basic_bounding_box {  
    public:  
        using data_type = typename GraphicsMath::bounding_box_data_type;  
  
        // 11.2.3, constructors:  
        basic_bounding_box() noexcept;  
        basic_bounding_box(float x, float y, float width, float height) noexcept;  
        basic_bounding_box(const basic_point_2d<GraphicsMath>& tl,  
                          const basic_point_2d<GraphicsMath>& br) noexcept;  
        basic_bounding_box(const data_type& val) noexcept;
```

```

// 11.2.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 11.2.5, modifiers:
void x(float val) noexcept;
void y(float val) noexcept;
void width(float val) noexcept;
void height(float val) noexcept;
void top_left(const basic_point_2d<GraphicsMath>& val) noexcept;
void bottom_right(const basic_point_2d<GraphicsMath>& val) noexcept;

// 11.2.6, observers:
float x() const noexcept;
float y() const noexcept;
float width() const noexcept;
float height() const noexcept;
basic_point_2d<GraphicsMath> top_left() const noexcept;
basic_point_2d<GraphicsMath> bottom_right() const noexcept;
};

// 11.2.7, operators:
template <class GraphicsMath>
bool operator==(const basic_bounding_box<GraphicsMath>& lhs,
               const basic_bounding_box<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator!=(const basic_bounding_box<GraphicsMath>& lhs,
               const basic_bounding_box<GraphicsMath>& rhs) noexcept;
}

```

11.2.3 basic_bounding_box constructors

[io2d.bounding_box.cons]

```
basic_bounding_box() noexcept;
```

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

2 *Postconditions:* `data() == GraphicsMath::create_bounding_box()`.

```
basic_bounding_box(float x, float y, float w, float h) noexcept;
```

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

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

5 *Postconditions:* `data() == GraphicsMath::create_bounding_box(x, y, w, h)`.

```
basic_bounding_box(const basic_point_2d<GraphicsMath>& tl,
                 const basic_point_2d<GraphicsMath>& br) noexcept;
```

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

7 *Postconditions:* `data() == GraphicsMath::create_bounding_box(tl.data(), br.data())`.

```
basic_bounding_box(const data_type& val) noexcept;
```

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

9 *Postconditions:* `data() == val`.

11.2.4 basic_bounding_box accessors

[io2d.bounding_box.accessors]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_bounding_box` object's data object (See: [11.2.1](#)).

11.2.5 basic_bounding_box modifiers

[io2d.bounding_box.modifiers]

```
void x(float v) noexcept;
```

1 *Effects:* Equivalent to `GraphicsMath::x(data(), v)`;

```

void y(float v) noexcept;
2   Effects: Equivalent to GraphicsMath::y(data(), v);

void width(float v) noexcept;
3   Effects: Equivalent to GraphicsMath::width(data(), v);

void height(float val) noexcept;
4   Effects: Equivalent to GraphicsMath::height(data(), v);

void top_left(const basic_point_2d<GraphicsMath>& v) noexcept;
5   Effects: Equivalent to GraphicsMath::top_left(data(), v.data());

void bottom_right(const basic_point_2d<GraphicsMath>& v) noexcept;
6   Effects: Equivalent to GraphicsMath::bottom_right(data(), v.data());

```

11.2.6 basic_bounding_box observers [io2d.bounding_box.observers]

```

float x() const noexcept;
1   Returns: GraphicsMath::x(data()).

float y() const noexcept;
2   Returns: GraphicsMath::y(data()).

float width() const noexcept;
3   Returns: GraphicsMath::width(data()).

float height() const noexcept;
4   Returns: GraphicsMath::height(data()).

basic_point_2d<GraphicsMath> top_left() const noexcept;
5   Returns: basic_point_2d<GraphicsMath>(GraphicsMath::top_left(data())).

basic_point_2d<GraphicsMath> bottom_right() const noexcept;
6   Returns: basic_point_2d<GraphicsMath>(GraphicsMath::bottom_right(data())).

```

11.2.7 basic_bounding_box operators [io2d.bounding_box.ops]

```

bool operator==(const basic_bounding_box<GraphicsMath>& lhs,
                const basic_bounding_box<GraphicsMath>& rhs) noexcept;
1   Returns: GraphicsMath::equal(lhs.data(), rhs.data()).

bool operator!=(const basic_bounding_box<GraphicsMath>& lhs,
                const basic_bounding_box<GraphicsMath>& rhs) noexcept;
2   Returns: GraphicsMath::not_equal(lhs.data(), rhs.data()).

```

11.3 Class basic_circle [io2d.circle]

11.3.1 basic_circle description [io2d.circle.intro]

- 1 The class template `basic_circle` describes a circle.
- 2 It has a *center* of type `basic_point_2d<GraphicsMath>` and a *radius* of type `float`.
- 3 The data are stored in an object of type `typename GraphicsMath::circle_data_type`. It is accessible using the data member functions.

11.3.2 basic_circle synopsis [io2d.circle.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsMath>
    class basic_circle {
    public:
        using data_type = typename GraphicsMath::circle_data_type;

```

```

// 11.3.3, constructors:
basic_circle() noexcept;
basic_circle(const basic_point_2d<GraphicsMath>& ctr, float rad) noexcept;
basic_circle(const typename GraphicsMath::circle_data_type& val) noexcept;

// 11.3.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 11.3.5, modifiers:
void center(const basic_point_2d<GraphicsMath>& ctr) noexcept;
void radius(float r) noexcept;

// 11.3.6, observers:
basic_point_2d<GraphicsMath> center() const noexcept;
float radius() const noexcept;
};

// 11.3.7, operators:
template <class GraphicsMath>
bool operator==(const basic_circle<GraphicsMath>& lhs,
  const basic_circle<GraphicsMath>& rhs) noexcept;
template <class GraphicsMath>
bool operator!=(const basic_circle<GraphicsMath>& lhs,
  const basic_circle<GraphicsMath>& rhs) noexcept;
}

```

11.3.3 basic_circle constructors

[io2d.circle.cons]

```
basic_circle() noexcept;
```

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

2 *Postconditions:* `data() == GraphicsMath::create_circle()`.

```
basic_circle(const basic_point_2d<GraphicsMath>& ctr, float r) noexcept;
```

Requires: `r >= 0.0f`.

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

4 *Postconditions:* `data() == GraphicsMath::create_circle(ctr, r)`.

11.3.4 basic_circle accessors

[io2d.circle.accessors]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_circle` object's data object (See: 11.3.1).

11.3.5 basic_circle modifiers

[io2d.circle.modifiers]

```
void center(const basic_point_2d<GraphicsMath>& ctr) noexcept;
```

1 *Effects:* Equivalent to `GraphicsMath::center(data(), ctr.data())`;

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

Requires: `r >= 0.0f`.

2 *Effects:* Equivalent to `GraphicsMath::radius(data(), r)`;

11.3.6 basic_circle observers

[io2d.circle.observers]

```
basic_point_2d<GraphicsMath> center() const noexcept;
```

1 *Returns:* `(basic_point_2d<GraphicsMath>(GraphicsMath::center(data())))`.

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

2 *Returns:* `GraphicsMath::radius(data())`.

11.3.7 basic_circle operators

[io2d.circle.ops]

```
bool operator==(const basic_circle<GraphicsMath>& lhs,  
               const basic_circle<GraphicsMath>& rhs) noexcept;
```

¹ *Returns:* GraphicsMath::equal(lhs.data(), rhs.data()).

```
bool operator!=(const basic_circle<GraphicsMath>& lhs,  
               const basic_circle<GraphicsMath>& rhs) noexcept;
```

² *Returns:* GraphicsMath::not_equal(lhs.data(), rhs.data()).

12 Text rendering and display [io2d.text]

12.1 Intro [io2d.text.intro]

- ¹ [*Note:* Measuring text and creating pre-formatted layouts will be coming in a future revision. — *end note*]
- ² For an overview of how text rendering works, see 16.3.8.
- ³ Text rendering and display makes use of the OFF Font Format. No other font formats are supported at this time.
- ⁴ References made to a font's tables refer to the tables contained in the font's OFF Font Format.
- ⁵ Text rendering and display uses UTF-8 encoded text contained in a `string` object. Attempting to use text in a different encoding results in undefined behavior.
- ⁶ Several enum classes and class template are used in the process of text rendering. These are defined here.

12.2 Enum class `font_size_units` [io2d.text.sizeunits]

12.2.1 `font_size_units` summary [io2d.text.sizeunits.summary]

- ¹ The `font_size_units` enum class specifies the measurement units used to interpret a font size value.
- ² Glyph data in a font is specified in *font size units*. The area in which the glyph data is defined is known as an *em box*, which is a two-dimensional square. The number of font size units that make up the length of a side of the em box, an *em*, is defined by the *unitsPerEm* value contained in the font's *head* table. Acceptable values for one em are between [16, 16384].
- ³ When a glyph is rendered, the font size unit coordinates are converted to ems by dividing 1 by em and multiplying the result by the coordinate value. This is then multiplied by the *interpreted font size* value to get the coordinates in surface space. The method for turning a font size value into an interpreted font size value is specified by the `font_size_units` enumerators.
- ⁴ The `font_size_units` enumerators

12.2.2 `font_size_units` synopsis [io2d.text.sizeunits.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_size_units {
        points,
        pixels
    };
}
```

12.2.3 `font_size_units` enumerators [io2d.text.sizeunits.enumerators]

Table 20 — `font_size_units` enumerator meanings

Enumerator	Meaning
<code>points</code>	The interpreted font size value $\frac{1}{6}$ of an inch. This relies on the PPI of the surface. The default value for a <code>basic_image_surface</code> is 96. This can be changed by calling the <code>ppi</code> member function of the <code>basic_image_surface</code> object. The default PPI values for other surface types is environment-specific and as such is unspecified. Users must query those surfaces to determine their PPI value.
<code>pixels</code>	The interpreted font size value is the same as the font size value.

- ¹ [*Note:* The PPI of output surfaces varies greatly depending on the device on which the program is running. Common values currently range anywhere from 96 to 300+. The value can even change when an output

surface is moved from one output device to another or when settings are changed on the existing output device. This is a difficult issue to deal with even when writing programs using environment-specific APIs. Using a `basic_image_surface` and setting its PPI to the current value of the output surface can help alleviate these problems, but if the PPI of the output surface changes, or if no adjustment from the default value is made, the resulting scaling will likely produce text that appears fuzzy or heavily pixelated depending on the parameters used when text rendering occurs. — *end note*]

12.3 Enum class `font_weight` [io2d.text.weight]

12.3.1 `font_weight` summary [io2d.text.weight.summary]

- ¹ The `font_weight` enum class indicates the visual weight (degree of blackness or thickness of strokes) of the characters in a font. The names of the enumerators correspond to the names of the *usWeightClass* values in the *OS/2* table described in the OFF Font Format and represent the same meaning as their counterparts in the OFF Font Format.
- ² The names of the enumerators substitute `_` for `-` in order to conform to C++ syntax.

12.3.2 `font_weight` synopsis [io2d.text.weight.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_weight {
        thin,
        extra_light,
        light,
        normal,
        medium,
        semi_bold,
        bold,
        extra_bold,
        black
    };
}
```

12.4 Enum class `font_capitalization` [io2d.text.caps]

12.4.1 `font_capitalization` summary [io2d.text.caps.summary]

- ¹ The `font_capitalization` enum class specifies that text should be rendered in a particular capitalization style. This value is ignored for text that is not in a bicameral script (i.e. those without case differences).
- ² Where the font that is being used provides small capital glyphs, glyph substitution shall be performed to replace lower case character glyphs with their small capital glyph counterparts as specified by the font.
- ³ Where the font that is being used does not provide separate small capital glyphs, small capitals shall be emulated by using uppercase character glyphs to replace their lowercase character glyph counterparts and then modifying the rendering of those replacement glyphs such that they are shorter than ordinary capital glyphs. The rendering method for shortening such glyphs is unspecified. In the case of emulation, it is recommended that the scaled height of the small capitals be the ratio of the *sxHeight* to the *sCapHeight* as specified in the *OS/2* table of the font.
- ⁴ [*Note:* To achieve a better look with emulation, in addition to scaling down the uppercase glyphs, using an increased weight (boldness) is often beneficial. — *end note*]

12.4.2 `font_capitalization` synopsis [io2d.text.caps.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_capitalization {
        normal,
        small_caps
    };
}
```

12.4.3 `font_capitalization` enumerators [io2d.text.caps.enumerators]

Table 21 — font_capitalization enumerator meanings

Enumerator	Meaning
normal	No change in rendering results from this enumerator value.
small_caps	The text is rendered with small capitals replacing lowercase characters.

12.5 Enum class font_stretching [io2d.text.stretching]

12.5.1 font_stretching summary [io2d.text.stretching.summary]

¹ The `font_stretching` enum class indicates a relative change from the normal aspect ratio (width to height ratio) as specified by a font designer for the glyphs in a font. The names of the enumerators correspond to the names of the *usWidthClass* values in the *OS/2* table described in the OFF Font Format and represent the same meaning as their counterparts in the OFF Font Format.

² The names of the enumerators substitute `_` for `-` in order to conform to C++ syntax.

12.5.2 font_stretching synopsis [io2d.text.stretching.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_stretching {
        ultra_condensed = 1,
        extra_condensed,
        condensed,
        semi_condensed,
        medium,
        normal = medium,
        semi_expanded,
        expanded,
        extra_expanded,
        ultra_expanded
    };
}
```

12.6 Enum class font_style [io2d.text.style]

12.6.1 font_style summary [io2d.text.style.summary]

¹ The `font_style` enum class specifies that a specific font pattern shall be used. If this font pattern is not available in the requested font family, a similar font family that contains a font face with this font pattern shall be used when creating the `basic_font` object.

12.6.2 font_style synopsis [io2d.text.style.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_style {
        normal,
        italic,
        oblique
    };
}
```

12.6.3 font_style enumerators [io2d.text.style.enumerators]

¹ All enumerators are defined in terms of bit flags set in the *fsSelection* value of the *OS/2* table of the font.

Table 22 — font_style enumerator meanings

Enumerator	Meaning
normal	A font face with bit 6 (REGULAR) set to 1.
italic	A font face with bit 0 (ITALIC) set to 1.
oblique	A font face with bit 9 (OBLIQUE) set to 1 or bit 1 (ITALIC) set to 1, with font rendering preference given to the font face that has bit 9 set to 1 if both font faces are present.

12.7 Enum class `font_line` [io2d.text.line]

12.7.1 `font_line` summary [io2d.text.line.summary]

- ¹ The `font_line` enum class specifies whether or not text should be underlined when rendered.

12.7.2 `font_line` synopsis [io2d.text.line.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_line {
        none,
        underline
    };
}
```

12.7.3 `font_line` enumerators [io2d.text.line.enumerators]

Table 23 — `font_line` enumerator meanings

Enumerator	Meaning
<code>none</code>	No underlining is performed when rendering text.
<code>underline</code>	Underlining is performed when rendering text.

12.8 Enum class `font_antialias` [io2d.text.antialias]

12.8.1 `font_antialias` summary [io2d.text.antialias.summary]

- ¹ The `font_antialias` enum class specifies whether or not text should be anti-aliased when rendered. Excluding the `font_antialias::none` enumerator, all enumerators specify preferences.
- ² Subpixel antialiasing takes advantage of the fact that most modern displays use pixels that have a red, a green, and a blue subcomponent. By manipulating which of these subcomponents are turned on for each pixel, the resulting text will appear to have less aliasing while retaining the intended color of the text as rendered when viewed by the user of the program.
- ³ Gray anti-aliasing uses varying shades of the color that the text is to be rendered with for pixels that are rendered and certain pixels that surround pixels that would be rendered if no anti-aliasing were performed in order to reduce aliasing. If a non-solid color brush is used, implementations may ignore this type of anti-aliasing even if they are otherwise capable of performing it.
- ⁴ [*Note:* With gray anti-aliasing, as examples, when the text is rendered as white, shades of gray would be used. When the text is rendered as green, shades of green would be used. — *end note*]
- ⁵ [*Note:* Anti-aliasing may not be available in certain environments, for certain font families, or in other circumstances, but it is always possible to not perform anti-aliasing. — *end note*]

12.8.2 `font_antialias` synopsis [io2d.text.antialias.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class font_antialias {
        none,
        antialias,
        gray,
        subpixel,
        prefer_gray,
        prefer_subpixel
    };
}
```

12.8.3 `font_antialias` enumerators [io2d.text.antialias.enumerators]

Table 24 — `font_antialias` enumerator meanings

Enumerator	Meaning
<code>none</code>	Do not anti-alias text when rendering

Table 24 — `font_antialias` enumerator meanings (continued)

Enumerator	Meaning
<code>antialias</code>	Prefer anti-aliasing, leaving it up to the implementation to decide on gray vs. subpixel.
<code>gray</code>	Use gray anti-aliasing if available, otherwise none.
<code>subpixel</code>	Use subpixel anti-aliasing if available, otherwise none.
<code>prefer_gray</code>	Prefer gray anti-aliasing if available, otherwise use subpixel if available.
<code>prefer_subpixel</code>	Prefer subpixel anti-aliasing if available, otherwise use gray if available.

12.9 Enum class `generic_font_names` [io2d.text.genericfonts]

12.9.1 `generic_font_names` summary [io2d.text.genericfonts.summary]

- ¹ The `generic_font_names` enum class specifies font names that correspond to *generic font families* found in the CSS Fonts Specification. A `basic_font` object created using a `generic_font_names` enumerator without specifying a font family through a `string` argument shall meet the requirements for a generic font family of the same name as specified in the CSS Fonts Specification.
- ² A `basic_font` object constructed using a `generic_font_names` enumerator shall return the name of the enumerator as a `string` object when its `family` member function is called.
- ³ [*Note:* As per the specification, multiple font families might be required to be used when using a `basic_font` object that is constructed using a `generic_font_names` enumerator in order to meet the specifications. Because of this, the `basic_font` object returns the enumerator name when its `family` member function is called rather than a more specific font family name. — *end note*]

12.9.2 `generic_font_names` synopsis [io2d.text.genericfonts.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class generic_font_names {
        serif,
        sans_serif,
        cursive,
        fantasy,
        monospace
    };
}
```

12.10 Class template `basic_text_props` [io2d.text.textprops]

12.10.1 `basic_text_props` summary [io2d.text.textprops.summary]

- ¹ The `basic_text_props` class template provides general state information that is applicable to the text rendering rendering and composing operations (16.3.2).
- ² It has a *scale* of type `float`, a *font size* of type `float`, a *size units type* of type `font_size_unit`, a *kerning value* of type `bool`, an *anti-aliasing value* of type `font_antialias`, a *stretch value* of type `font_stretching`, a *strike through value* of type `bool`, and a *font line value* of type `font_line`.
- ³ The data are stored in an object of type `typename GraphicsSurfaces::text::text_props_data_type`. It is accessible using the `data` member function.

12.10.2 `basic_text_props` synopsis [io2d.text.textprops.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_text_props {
    public:
        using data_type = typename GraphicsSurfaces::text::text_props_data_type;
    };
}
```

```

// 12.10.3, construct:
basic_text_props(float scl = 1.0f,
    font_size_units fsu = font_size_units::points,
    float sz = -1.0f,
    bool kern = true,
    font_antialias aa = font_antialias::antialias,
    font_stretching stretch = font_stretching::normal,
    bool strike = false,
    font_line fl = font_line::none) noexcept;

// 12.10.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 12.10.5, modifiers:
void scale(float s) noexcept;
void font_size(font_size_units fsu, float sz) noexcept;
void kerning(bool k) noexcept;
void antialiasing(font_antialias aa) noexcept;
void stretching(font_stretching fs) noexcept;
void strike_through(bool st) noexcept;
void line(font_line fl) noexcept;

// 12.10.6, observers:
float scale() const noexcept;
float font_size() const noexcept;
font_size_units size_units() const noexcept;
bool kerning() const noexcept;
font_antialias antialiasing() const noexcept;
font_stretching stretching() const noexcept;
bool strike_through() const noexcept;
font_line line() const noexcept;
};
}

```

12.10.3 basic_text_props constructor

[io2d.text.textprops.ctor]

```

basic_text_props(float scl = 1.0f,
    font_size_units fsu = font_size_units::points,
    float sz = -1.0f,
    bool kern = true,
    font_antialias aa = font_antialias::antialias,
    font_stretching stretch = font_stretching::normal,
    bool strike = false,
    font_line fl = font_line::none) noexcept;

```

- 1 *Effects:* Constructs an object of type `basic_text_props`.
- 2 *Postconditions:* `data() == GraphicsSurfaces::text::create_text_props(scl, fsu, sz, kern, aa, stretch, strike, fl)`.
- 3 The scale is `scl`. The size units type is `fsu`. The font size is `sz`. The kerning value is `kern`. The anti-aliasing value is `aa`. The stretch value is `stretch`. The strike through value is `strike`. The font line value is `fl`.

12.10.4 Accessors

[io2d.text.textprops.acc]

```

const data_type& data() const noexcept;
data_type& data() noexcept;

```

- 1 *Returns:* A reference to the `basic_text_props` object's data object (See: [17.2.5.1](#)).
- 2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

12.10.5 basic_text_props modifiers**[io2d.text.textprops.mod]**

```

void scale(float scl) noexcept;
1   Effects: Calls GraphicsSurfaces::text::scale(data(), scl).
2   Remarks: The scale is scl.
3   If the scale is  $\leq 0$ , the scale is disregarded during text rendering occurs.

void font_size(font_size_units fsu, float sz) noexcept;
4   Effects: Calls GraphicsSurfaces::text::font_size(data(), fsu, sz).
5   Remarks: The size units type is fsu. The font size is sz.
6   If the font size is leq0, the basic_font object's font size is used during text rendering.

void kerning(bool k) noexcept;
7   Effects: Calls GraphicsSurfaces::text::kerning(data(), k).
8   Remarks: The kerning value is k.
9   When the kerning value is true, kerning should be used.
10  [Note: If kerning is not available with the font being used, kerning does not occur. — end note]

void antialiasing(font_antialias aa) noexcept;
11  Effects: Calls GraphicsSurfaces::text::antialiasing(data(), aa).
12  Remarks: The anti-aliasing value is aa.

void stretching(font_stretching fs) noexcept;
13  Effects: Calls GraphicsSurfaces::text::stretching(data(), fs).
14  Remarks: The stretch value is fs.

void strike_through(bool st) noexcept;
15  Effects: Calls GraphicsSurfaces::text::strike_through(data(), st).
16  Remarks: The strike through value is st.

void line(font_line fl) noexcept;
17  Effects: Calls GraphicsSurfaces::text::line(data(), fl).
18  Remarks: The font line value is fl.

```

12.10.6 basic_text_props observers**[io2d.text.textprops.obs]**

```

float scale() const noexcept;
1   Returns: GraphicsSurfaces::text::scale(data()).
2   Remarks: The returned value is the scale.

float font_size() const noexcept;
3   Returns: GraphicsSurfaces::text::font_size(data()).
4   Remarks: The returned value is the font size.

font_size_units size_units() const noexcept;
5   Returns: GraphicsSurfaces::text::size_units(data()).
6   Remarks: The returned value is the size units type.

bool kerning() const noexcept;
7   Returns: GraphicsSurfaces::text::kerning(data()).
8   Remarks: The returned value is the kerning value.

font_antialias antialiasing() const noexcept;
9   Returns: GraphicsSurfaces::text::antialiasing(data()).

```

10 *Remarks:* The returned value is the anti-aliasing value.

```
font_stretching stretching() const noexcept;
```

11 *Returns:* GraphicsSurfaces::text::stretching(data()).

12 *Remarks:* The returned value is the stretch value.

```
bool strike_through() const noexcept;
```

13 *Returns:* GraphicsSurfaces::text::strike_through(data()).

14 *Remarks:* The returned value is the strike through value.

```
font_line line() const noexcept;
```

15 *Returns:* GraphicsSurfaces::text::line(data()).

16 *Remarks:* The returned value is the font line value.

12.11 Class template `basic_font` [io2d.text.font]

12.11.1 `basic_font` summary [io2d.text.font.summary]

- 1 The `basic_font` class template provides an OFF Font Format font for use by the text rendering rendering and composing operation (16.3.8).
- 2 It has a *font family* of type `string`, a *font size* of type `float`, a *size units type* of type `font_size_unit`, a *font weight* of type `font_weight`, a *font style* of type `font_style`, and a *merging* value of `bool`.
- 3 The data are stored in an object of type `typename GraphicsSurfaces::text::font_data_type`. It is accessible using the `data` member function.

12.11.2 `basic_font` matching [io2d.text.font.matching]

- 1 Sometimes when creating a `basic_font` object, the font family that is requested is not present in the environment the program is running in. When this occurs, the implementation uses the other parameters, together with its knowledge, if any, of the requested family to select a similar font family to the one requested. The selection algorithm is unspecified. When this occurs, the `family` member function shall return the name of the font the implementation selected.

12.11.3 `basic_font` synopsis [io2d.text.font.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_font {
    public:
        using data_type = typename GraphicsSurfaces::text::font_data_type;

        // 12.11.4, construct:
        basic_font(string family, font_size_units fsu, float sz,
            generic_font_names gfn, font_weight fw = font_weight::normal,
            font_style fs = font_style::normal, bool merging = true);
        basic_font(filesystem::path file, string family, font_size_units fsu,
            float sz, font_weight fw = font_weight::normal, font_style fs =
            font_style::normal, bool merging = true);
        basic_font(generic_font_names gfn, font_size_units fsu, float sz,
            font_weight fw = font_weight::normal, font_style fs = font_style::normal);

        // 12.11.5, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 12.11.6, modifiers:
        void font_size(font_size_units fsu, float sz) noexcept;
        void merging(bool m) noexcept;

        // 12.11.7, observers:
        float font_size() const noexcept;
        string family() const noexcept;
    };
};
```

```

    font_size_units size_units() const noexcept;
    font_weight weight() const noexcept;
    font_style style() const noexcept;
    bool merging() const noexcept;
};
}

```

12.11.4 basic_font constructor

[io2d.text.font.ctor]

```

basic_font(string family, font_size_units fsu, float sz,
    generic_font_names gfn, font_weight fw = font_weight::normal,
    font_style fs = font_style::normal, bool merging = true);

```

1 *Requires:* `sz > 0`.

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

3 *Postconditions:* `data() == GraphicsSurfaces::text::create_font(family, fsu, sz, gfn, fw, fs, merging)`.

4 *Remarks:* The value of `gfn` is only used if an exact match is not available (see: [12.11.2](#)).

5 If an exact match is available, the font family is `family`. The size units type is `fsu`. The font size is `sz`. The font weight is `fw`. The font style is `fs`. The stretch value is `stretch`. The strike through value is `strike`. The font line value is `fl`. The merging value is `merging`.

6 When an exact match is not available, the implementation uses the value of `gfn` to help select a similar font. In this case, the font family is the name of the font chosen by the implementation. The size units type is `fsu`. The font size is `sz`. The font weight is the font weight of the font chosen by the implementation. The font style is the font style of the font chosen by the implementation. The stretch value is `stretch`. The strike through value is `strike`. The font line value is `fl`. The merging value is `merging`.

```

basic_font(filesystem::path file, string family, font_size_units fsu, float sz,
    font_weight fw = font_weight::normal, font_style fs = font_style::normal,
    bool merging = true);

```

7 *Requires:* `sz > 0`.

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

9 *Postconditions:* `data() == GraphicsSurfaces::text::create_font(file, family, fsu, sz, fw, fs, merging)`.

10 *Remarks:* The font family is specified in the `name` table of the OFF Font Format font contained in `file`. If the OFF Font Format font contained in `file` is an OFF Font Font Collection, the font family is `family` if a font with the family name `family` is contained in `file`, otherwise it is the family name of the font contained in that file selected by the implementation in an unspecified manner. The size units type is `fsu`. The font size is `sz`. The font weight is the font weight of the font in the OFF Font Format font contained in `file`, or is the nearest font weight to the value specified by `fw` if the `file` is an OFF Font Format Font Collection. The font style is the font style of the font in the OFF Font Format font contained in `file`, or is the nearest font style to the value of `fs` if `file` is an OFF Font Format Font Collection. The stretch value is `stretch`. The strike through value is `strike`. The font line value is `fl`. The merging value is `merging`.

```

basic_font(generic_font_names gfn, font_size_units fsu, float sz,
    font_weight fw = font_weight::normal, font_style fs = font_style::normal);

```

11 *Requires:* `sz > 0`.

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

13 *Postconditions:* `data() == GraphicsSurfaces::text::create_font(gfn, fsu, sz, fw, fs)`.

14 *Remarks:* The font is chosen by the implementation using `gfn` together with the other arguments to perform matching (see [12.11.2](#)).

15 The font family is the string that is the name of the enumerator value specified by `gfn`. The font is chosen by the implementation as described in `io2d.text.genericfonts.summary`, taking into account the values of `fw` and `fs`. The size units type is `fsu`. The font size is `sz`. The font weight is the font

weight of the font chosen by the implementation. The font style is the font style of the font chosen by the implementation. The stretch value is `stretch`. The strike through value is `strike`. The font line value is `fl`. The merging value is `true`.

16 [*Note*: The merging value is true because the intention of this constructor is to support as many characters as possible, in the manner specified by the CSS Fonts Specification. The user is mostly surrendering control of the choice of a font family in exchange for this expansive character support. It can be changed using the `merging` member function, but this is strongly discouraged. — *end note*]

12.11.5 Accessors

[io2d.text.font.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns*: A reference to the `basic_font` object's data object (See: [17.2.5.1](#)).

2 *Remarks*: The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

12.11.6 `basic_font` modifiers

[io2d.text.font.mod]

```
void font_size(font_size_units fsu, float sz) noexcept;
```

1 *Requires*: `sz > 0`.

2 *Effects*: Calls `GraphicsSurfaces::text::font_size(data(), fsu, sz)`.

3 *Remarks*: The size units type is `fsu`. The font size is `sz`.

```
void merging(bool m) noexcept;
```

4 *Effects*: Calls `GraphicsSurfaces::text::merging(data(), m)`.

5 *Remarks*: The merging value is `m`.

12.11.7 `basic_font` observers

[io2d.text.font.obs]

```
float font_size() const noexcept;
```

1 *Returns*: `GraphicsSurfaces::text::font_size(data())`.

2 *Remarks*: The returned value is the font size.

```
font_size_units size_units() const noexcept;
```

3 *Returns*: `GraphicsSurfaces::text::size_units(data())`.

4 *Remarks*: The returned value is the size units type.

```
string family() const noexcept;
```

5 *Returns*: `GraphicsSurfaces::text::family(data())`.

6 *Remarks*: The returned value is the font family. If matching occurred (see: [12.11.2](#)), the value is the name of the family that the implementation chose.

```
font_weight weight() const noexcept;
```

7 *Returns*: `GraphicsSurfaces::text::weight(data())`.

8 *Remarks*: The returned value is the font weight.

```
font_style style() const noexcept;
```

9 *Returns*: `GraphicsSurfaces::text::style(data())`.

10 *Remarks*: The returned value is the font style.

```
bool merging() const noexcept;
```

11 *Returns*: `GraphicsSurfaces::text::merging(data())`.

12 *Remarks*: The returned value is the merging value.

12.12 Class template `basic_font_database` [io2d.text.fontdb]

12.12.1 `basic_font_database` summary [io2d.text.fontdb.summary]

- ¹ The `basic_font_database` class template provides a way to get a list of all font families available in the environment the program is running in.
- ² It has no public data.
- ³ The data are stored in an object of type `typename GraphicsSurfaces::text::font_database_data_type`. It is accessible using the `data` member function.

12.12.2 `basic_font_database` matching [io2d.text.fontdb.matching]

12.12.3 `basic_font_database` synopsis [io2d.text.fontdb.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_font_database {
    public:
        using data_type = typename GraphicsSurfaces::text::font_database_data_type;

        // 12.12.4, construct:
        basic_font_database();

        // 12.12.5, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 12.12.6, observers:
        vector<string> get_families() const noexcept;
    };
}
```

12.12.4 `basic_font_database` constructor [io2d.text.fontdb.ctor]

```
basic_font_database();
```

- ¹ *Effects:* Constructs an object of type `basic_font_database`.
- ² *Postconditions:* `data() == GraphicsSurfaces::text::create_font_database()`.

12.12.5 Accessors [io2d.text.fontdb.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

- ¹ *Returns:* A reference to the `basic_font_database` object's data object (See: [17.2.5.1](#)).
- ² *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

12.12.6 `basic_font_database` observers [io2d.text.fontdb.obs]

```
vector<string> get_families() const noexcept;
```

- ¹ *Returns:* `GraphicsSurfaces::text::get_families(data())`.
- ² *Remarks:* The returned value is a collection containing the names of all font families available in the environment the program is running in.

13 Paths

[io2d.paths]

13.1 Overview of paths

[io2d.paths.overview]

- ¹ Paths define geometric objects which can be stroked (Table 41), filled, and used to define a clip area (See: 15.11.1).
- ² A path contains zero or more figures.
- ³ A figure is composed of at least one segment.
- ⁴ A figure may contain degenerate segments. When a path is interpreted (13.3.16), degenerate segments are removed from figures. [*Note*: If a path command exists or is inserted between segments, it's possible that points which might have compared equal will no longer compare equal as a result of interpretation (13.3.16). — *end note*]
- ⁵ Paths provide vector graphics functionality. As such they are particularly useful in situations where an application is intended to run on a variety of platforms whose output devices (16.3.9) span a large gamut of sizes, both in terms of measurement units and in terms of a horizontal and vertical pixel count, in that order.
- ⁶ A `basic_interpreted_path` object is an immutable resource wrapper containing a path (13.4). A `basic_interpreted_path` object is created by interpreting the path contained in a `basic_path_builder` object. It can also be default constructed, in which case the `basic_interpreted_path` object contains no figures. [*Note*: `basic_interpreted_path` objects provide significant optimization opportunities for implementations. Because they are immutable and opaque, they are intended to be used to store a path in the most efficient representation available. — *end note*]

13.2 Path examples (Informative)

[io2d.paths.example]

13.2.1 Overview

[io2d.paths.example.intro]

- ¹ Paths are composed of zero or more figures. The following examples show the basics of how paths work in practice.
- ² Every example is placed within the following code at the indicated spot. This code is shown here once to avoid repetition:

```
#include <experimental/io2d>

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

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

    // Example code goes here.

    // Example code ends.

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

13.2.2 Example 1

[io2d.paths.examples.one]

- ¹ Example 1 consists of a single figure, forming a trapezoid:

```
pb.new_figure({ 80.0f, 20.0f }); // Begins the figure.
pb.line({ 220.0f, 20.0f }); // Creates a line from the [80, 20] to [220, 20].
pb.rel_line({ 60.0f, 160.0f }); // Line from [220, 20] to
```

```

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

```

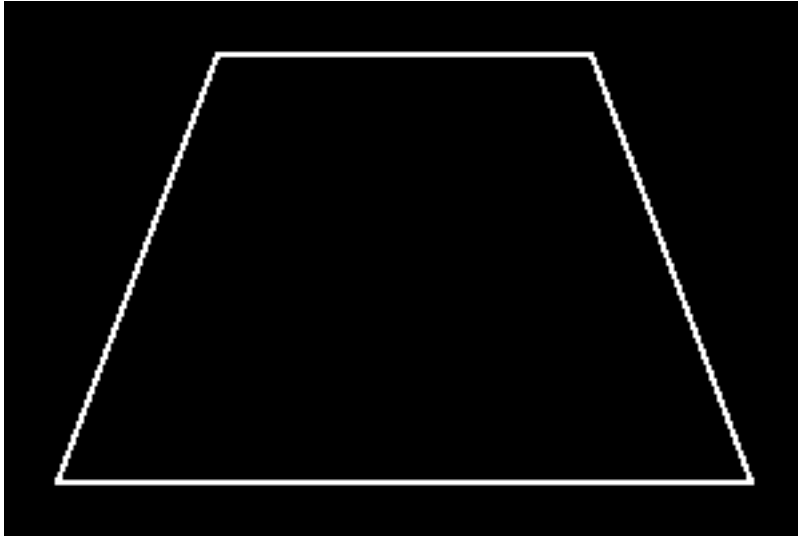


Figure 1 — Example 1 result

13.2.3 Example 2

[io2d.paths.examples.two]

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

```

    pb.new_figure({ 20.0f, 20.0f }); // Begin the first figure.
    pb.rel_line({ 100.0f, 0.0f });
    pb.rel_line({ 0.0f, 160.0f });
    pb.rel_line({ -100.0f, 0.0f });
    pb.rel_line({ 0.0f, -160.0f });

    pb.new_figure({ 180.0f, 20.0f }); // End the first figure and begin the
    // second figure.

    pb.rel_line({ 100.0f, 0.0f });
    pb.rel_line({ 0.0f, 160.0f });
    pb.rel_line({ -100.0f, 0.0f });
    pb.close_figure(); // End the second figure.
    imgSfc.stroke(foreBrush, pb, nullopt, stroke_props{ 10.0f }, nullopt,
    aliased);

```

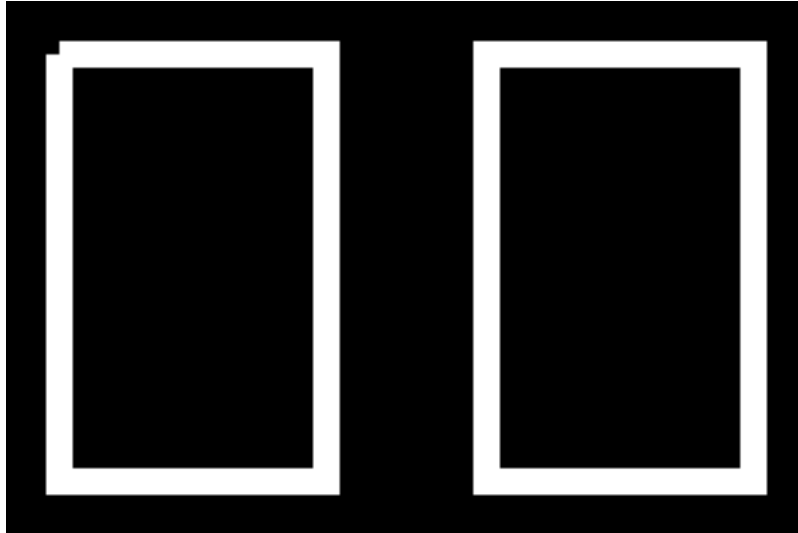


Figure 2 — Example 2 result

- 2 The resulting image from example 2 shows the difference between an open figure and a closed figure. Each figure begins and ends at the same point. The difference is that with the closed figure, that the rendering of the point where the initial segment and final segment meet is controlled by the `line_join` value in the `stroke_props` class, which in this case is the default value of `line_join::miter`. In the open figure, the rendering of that point receives no special treatment such that each segment at that point is rendered using the `line_cap` value in the `stroke_props` class, which in this case is the default value of `line_cap::none`.
- 3 That difference between rendering as a `line_join` versus rendering as two `line_caps` is what causes the notch to appear in the open segment. Segments are rendered such that half of the stroke width is rendered on each side of the point being evaluated. With no line cap, each segment begins and ends exactly at the point specified.
- 4 So for the open figure, the first line begins at `point_2d{ 20.0f, 20.0f }` and the last line ends there. Given the stroke width of `10.0f`, the visible result for the first line is a rectangle with an upper left corner of `point_2d{ 20.0f, 15.0f }` and a lower right corner of `point_2d{ 120.0f, 25.0f }`. The last line appears as a rectangle with an upper left corner of `point_2d{ 15.0f, 20.0f }` and a lower right corner of `point_2d{ 25.0f, 180.0f }`. This produces the appearance of a square gap between `point_2d{ 15.0f, 15.0f }` and `point_2d{20.0f, 20.0f }`.
- 5 For the closed figure, adjusting for the coordinate differences, the rendering facts are the same as for the open figure except for one key difference: the point where the first line and last line meet is rendered as a line join rather than two line caps, which, given the default value of `line_join::miter`, produces a miter, adding that square area to the rendering result.

13.2.4 Example 3

[io2d.paths.examples.three]

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

```
pb.new_figure({ 20.0f, 20.0f });
pb.rel_quadratic_curve({ 60.0f, 120.0f }, { 60.0f, -120.0f });
pb.rel_new_figure({ 20.0f, 0.0f });
pb.rel_quadratic_curve({ 60.0f, 120.0f }, { 60.0f, -120.0f });
pb.close_figure();
pb.new_figure({ 20.0f, 150.0f });
pb.rel_cubic_curve({ 40.0f, -120.0f }, { 40.0f, 120.0f * 2.0f },
  { 40.0f, -120.0f });
pb.rel_new_figure({ 20.0f, 0.0f });
pb.rel_cubic_curve({ 40.0f, -120.0f }, { 40.0f, 120.0f * 2.0f },
  { 40.0f, -120.0f });
pb.close_figure();
imgSfc.stroke(foreBrush, pb, nullopt, nullopt, nullopt, aliased);
```

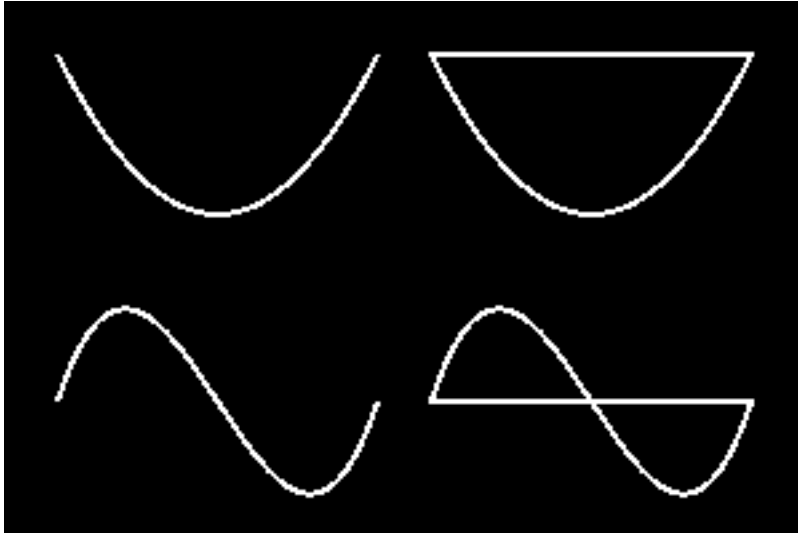


Figure 3 — Path example 3

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

13.2.5 Example 4

[`io2d.paths.examples.four`]

- ¹ Example 4 shows how to draw "C++" using figures.
- ² For the "C", it is created using an arc. A scaling matrix is used to make it slightly elliptical. It is also desirable that the arc has a fixed center point, `point_2d{ 85.0f, 100.0f }`. The inverse of the scaling matrix is used in combination with the `point_for_angle` function to determine the point at which the arc should begin in order to get achieve this fixed center point. The "C" is then stroked.
- ³ Unlike the "C", which is created using an open figure that is stroked, each "+" is created using a closed figure that is filled. To avoid filling the "C", `pb.clear()`; is called to empty the container. The first "+" is created using a series of lines and is then filled.
- ⁴ Taking advantage of the fact that `path_builder` is a container, rather than create a brand new figure for the second "+", a translation matrix is applied by inserting a `figure_items::change_matrix` figure item before the `figure_items::new_figure` object in the existing plus, reverting back to the old matrix immediately after the and then filling it again.

```
// Create the "C".
const matrix_2d scl = matrix_2d::init_scale({ 0.9f, 1.1f });
auto pt = scl.inverse().transform_pt({ 85.0f, 100.0f }) +
  point_for_angle(half_pi<float> / 2.0f, 50.0f);
pb.matrix(scl);
pb.new_figure(pt);
pb.arc({ 50.0f, 50.0f }, three_pi_over_two<float>, half_pi<float> / 2.0f);
imgSfc.stroke(foreBrush, pb, nullopt, stroke_props{ 10.0f });
// Create the first "+".
pb.clear();
pb.new_figure({ 130.0f, 105.0f });
pb.rel_line({ 0.0f, -10.0f });
pb.rel_line({ 25.0f, 0.0f });
pb.rel_line({ 0.0f, -25.0f });
pb.rel_line({ 10.0f, 0.0f });
pb.rel_line({ 0.0f, 25.0f });
pb.rel_line({ 25.0f, 0.0f });
pb.rel_line({ 0.0f, 10.0f });
pb.rel_line({ -25.0f, 0.0f });
pb.rel_line({ 0.0f, 25.0f });
pb.rel_line({ -10.0f, 0.0f });
pb.rel_line({ 0.0f, -25.0f });
```

```

pb.close_figure();
imgSfc.fill(foreBrush, pb);
// Create the second "+".
pb.insert(pb.begin(), figure_items::change_matrix(
    matrix_2d::init_translate({ 80.0f, 0.0f })));
imgSfc.fill(foreBrush, pb);

```

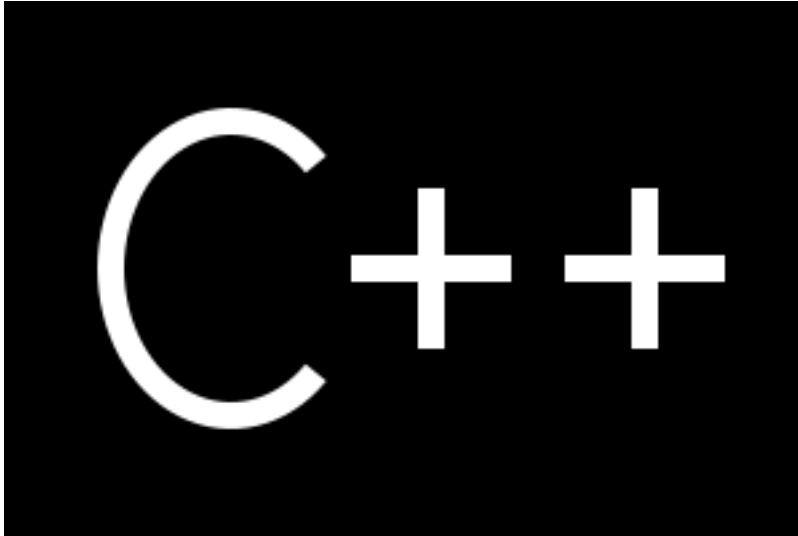


Figure 4 — Path example 4

13.3 Class template `basic_figure_items` [io2d.paths.figureitems]

13.3.1 Introduction [io2d.paths.figureitems.intro]

- ¹ The nested classes within the class template `basic_figure_items` describe figure items.
- ² A figure begins with an `abs_new_figure` or `rel_new_figure` object. A figure ends when:
 - (2.1) — a `close_figure` object is encountered;
 - (2.2) — a `abs_new_figure` or `rel_new_figure` object is encountered; or
 - (2.3) — there are no more figure items in the path.
- ³ The `basic_path_builder` class template is a sequential container that contains a path. It provides a simple interface for building a path. A path can also be created using any container that stores `basic_figure_items<GraphicsSurfaces>::figure_item` objects.

13.3.2 Synopsis [io2d.paths.figureitems.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items {
    public:
        class abs_new_figure;
        class rel_new_figure;
        class close_figure;
        class abs_matrix;
        class rel_matrix;
        class revert_matrix;
        class abs_cubic_curve;
        class rel_cubic_curve;
        class abs_line;
        class rel_line;
        class abs_quadratic_curve;
        class rel_quadratic_curve;
        class arc;
    };
}

```

```

using figure_item = variant<abs_new_figure, rel_new_figure,
    close_figure, abs_matrix, rel_matrix, revert_matrix, abs_cubic_curve,
    rel_cubic_curve, abs_line, rel_line, abs_quadratic_curve,
    rel_quadratic_curve, arc>;
};

```

13.3.3 Class template `basic_figure_items<GraphicsSurfaces>::abs_new_figure` `[io2d.absnewfigure]`

13.3.3.1 Overview `[io2d.absnewfigure.intro]`

- ¹ The class template `basic_figure_items<GraphicsSurfaces>::abs_new_figure` describes a figure item that is a new figure command.
- ² It has an *at point* of type `basic_point_2d<GraphicsSurfaces::graphics_math_type>`.
- ³ The data are stored in an object of type `typename GraphicsSurfaces::paths::abs_new_figure_data_type`. It is accessible using the data member functions.

13.3.3.2 Synopsis `[io2d.absnewfigure.synopsis]`

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::abs_new_figure {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::abs_new_figure_data_type;

        // 13.3.3.3, construct:
        abs_new_figure();
        explicit abs_new_figure(const basic_point_2d<graphics_math_type>& pt);
        abs_new_figure(const abs_new_figure& other) = default;
        abs_new_figure(abs_new_figure&& other) noexcept = default;

        // assign:
        abs_new_figure& operator=(const abs_new_figure& other) = default;
        abs_new_figure& operator=(abs_new_figure&& other) noexcept = default;

        // 13.3.3.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.3.5, modifiers:
        void at(const basic_point_2d<graphics_math_type>& pt) noexcept;

        // 13.3.3.6, observers:
        basic_point_2d<graphics_math_type> at() const noexcept;
    };

    // 13.3.3.7, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& rhs)
        noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& rhs)
        noexcept;
}

```

13.3.3.3 Constructors `[io2d.absnewfigure.ctor]`

```
abs_new_figure();
```

- ¹ *Effects:* Constructs an object of type `abs_new_figure`.

2 *Postconditions:* `data() == GraphicsSurfaces::paths::create_abs_new_figure()`.

3 *Remarks:* The `at` point is `basic_point_2d<graphics_math_type>()`.

```
explicit abs_new_figure(const basic_point_2d<graphics_math_type>& pt);
```

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

5 *Postconditions:* `data() == GraphicsSurfaces::paths::create_abs_new_figure(pt)`.

6 *Remarks:* The `at` point is `pt`.

13.3.3.4 Accessors

[io2d.absnewfigure.acc]

```
const data_type& data() const noexcept;
```

```
data_type& data() noexcept;
```

1 *Returns:* A reference to the `abs_new_figure` object's data object (See: 13.3.3.1).

13.3.3.5 Modifiers

[io2d.absnewfigure.mod]

```
void at(const basic_point_2d<graphics_math_type>& pt) noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::paths::at(data(), pt)`.

2 *Remarks:* The `at` point is `pt`.

13.3.3.6 Observers

[io2d.absnewfigure.obs]

```
basic_point_2d<graphics_math_type> at() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::paths::at(data())`.

2 *Remarks:* The returned value is the `at` point.

13.3.3.7 Equality operators

[io2d.absnewfigure.eq]

```
template <class GraphicsSurfaces>
```

```
bool operator==(
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& lhs,
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& rhs)
```

```
    noexcept;
```

1 *Returns:* `GraphicsSurfaces::paths::equal(lhs, rhs)`.

```
template <class GraphicsSurfaces>
```

```
bool operator!=(
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& lhs,
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_new_figure& rhs)
```

```
    noexcept;
```

2 *Returns:* `GraphicsSurfaces::paths::not_equal(lhs, rhs)`.

13.3.4 Class template `basic_figure_items<GraphicsSurfaces>::rel_new_figure`

[io2d.relnfigure]

13.3.4.1 Overview

[io2d.relnfigure.intro]

1 The class template `basic_figure_items<GraphicsSurfaces>::rel_new_figure` describes a figure item that is a new figure command.

2 It has an *at point* of type `basic_point_2d<GraphicsSurfaces::graphics_math_type>`.

3 The data are stored in an object of type `typename GraphicsSurfaces::paths::rel_new_figure_data_`-type. It is accessible using the data member functions.

13.3.4.2 Synopsis

[io2d.relnfigure.synopsis]

```
namespace std::experimental::io2d::v1 {
```

```
    template <class GraphicsSurfaces>
```

```
    class basic_figure_items<GraphicsSurfaces>::rel_new_figure {
```

```
    public:
```

```
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
```

```
        using data_type =
```

```
            typename GraphicsSurfaces::paths::rel_new_figure_data_type;
```

```

// 13.3.4.3, construct:
rel_new_figure();
explicit rel_new_figure(const basic_point_2d<graphics_math_type>& pt);
rel_new_figure(const rel_new_figure& other) = default;
rel_new_figure(rel_new_figure&& other) noexcept = default;

// assign:
rel_new_figure& operator=(const rel_new_figure& other) = default;
rel_new_figure& operator=(rel_new_figure&& other) noexcept = default;

// 13.3.4.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 13.3.4.5, modifiers:
void at(const basic_point_2d<graphics_math_type>& pt) noexcept;

// 13.3.4.6, observers:
basic_point_2d<graphics_math_type> at() const noexcept;
};

// 13.3.4.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& rhs)
    noexcept;
}

```

13.3.4.3 Constructors

[io2d.relnewfigure.ctor]

```
rel_new_figure() noexcept;
```

- 1 *Effects:* Constructs an object of type `rel_new_figure`.
- 2 *Postconditions:* `data() == GraphicsSurfaces::paths::create_rel_new_figure()`.
- 3 *Remarks:* The at point is `basic_point_2d<graphics_math_type>()`.

```
explicit rel_new_figure(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt) noexcept;
```

- 4 *Effects:* Constructs an object of type `rel_new_figure`.
- 5 *Postconditions:* `data() == GraphicsSurfaces::paths::create_rel_new_figure(pt)`.
- 6 *Remarks:* The at point is `pt`.

13.3.4.4 Accessors

[io2d.relnewfigure.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

- 1 *Returns:* A reference to the `rel_new_figure` object's data object (See: 13.3.4.1).

13.3.4.5 Modifiers

[io2d.relnewfigure.mod]

```
void at(const basic_point_2d<graphics_math_type>& pt) noexcept;
```

- 1 *Effects:* Calls `GraphicsSurfaces::paths::at(data(), pt)`.
- 2 *Remarks:* The at point is `pt`.

13.3.4.6 Observers

[io2d.renewfigure.obs]

```
basic_point_2d<graphics_math_type> at() const noexcept;
```

1 *Returns:* GraphicsSurfaces::paths::at(data()).

2 *Remarks:* The returned value is the at point.

13.3.4.7 Equality operators

[io2d.renewfigure.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& rhs)
noexcept;
```

1 *Returns:* GraphicsSurfaces::paths::equal(lhs, rhs).

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_new_figure& rhs)
noexcept;
```

2 *Returns:* GraphicsSurfaces::paths::not_equal(lhs, rhs).

13.3.5 Class template basic_figure_items<GraphicsSurfaces>::close_figure

[io2d.closefigure]

13.3.5.1 Overview

[io2d.closefigure.intro]

1 The class template basic_figure_items<GraphicsSurfaces>::close_figure describes a figure item that is a close figure command.

13.3.5.2 Synopsis

[io2d.closefigure.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::close_figure {
    public:
        // construct:
        close_figure() = default;
        close_figure(const close_figure& other) = default;
        close_figure(close_figure&& other) noexcept = default;

        // assign:
        close_figure& operator=(const close_figure& other) = default;
        close_figure& operator=(close_figure&& other) noexcept = default;
    };

    // 13.3.5.3, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::close_figure& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::close_figure& rhs)
    noexcept;

    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::close_figure& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::close_figure& rhs)
    noexcept;
}
```

13.3.5.3 Equality operators

[io2d.closefigure.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& rhs)
```

```
noexcept;
```

1 *Returns:* true.

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::close_figure& rhs)
noexcept;
```

2 *Returns:* false.

13.3.6 Class template `basic_figure_items<GraphicsSurfaces>::abs_matrix` [io2d.absmatrix]

13.3.6.1 Overview

[io2d.absmatrix.intro]

- 1 The class template `basic_figure_items<GraphicsSurfaces>::abs_matrix` describes a figure item that is a path command.
- 2 It has a transform matrix of type `basic_matrix_2d<GraphicsSurfaces::graphics_math_type>`.
- 3 The data are stored in an object of type `typename GraphicsSurfaces::paths::abs_matrix_data_type`. It is accessible using the data member functions.

13.3.6.2 Synopsis

[io2d.absmatrix.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::abs_matrix {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::abs_matrix_data_type;

        // 13.3.6.3, construct:
        abs_matrix();
        explicit abs_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
        abs_matrix(const abs_matrix& other) = default;
        abs_matrix(abs_matrix&& other) noexcept = default;

        // assign:
        abs_matrix& operator=(const abs_matrix& other) = default;
        abs_matrix& operator=(abs_matrix&& other) noexcept = default;

        // 13.3.6.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.6.5, modifiers:
        void matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;

        // 13.3.6.6, observers:
        basic_matrix_2d<graphics_math_type> matrix() const noexcept;
    };

    // 13.3.6.7, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& rhs)
    noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& rhs)
    noexcept;
}
```

13.3.6.3 Constructors

[io2d.absmatrix.ctor]

```
abs_matrix() noexcept;
```

1 *Effects:* Constructs an `abs_matrix` object.

2 *Postconditions:* `data() == GraphicsSurfaces::paths::create_abs_matrix()`.

3 *Remarks:* The transform matrix is `basic_matrix_2d<graphics_math_type>()`.

```
explicit abs_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
```

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

5 *Effects:* Constructs an `abs_matrix` object.

6 *Postconditions:* `data() == GraphicsSurfaces::paths::create_abs_matrix(m)`.

7 *Remarks:* The transform matrix is `m`.

13.3.6.4 Accessors

[io2d.absmatrix.acc]

```
const data_type& data() const noexcept;
```

```
data_type& data() noexcept;
```

1 *Returns:* A reference to the `abs_matrix` object's data object (See: 13.3.6.1).

13.3.6.5 Modifiers

[io2d.absmatrix.mod]

```
void matrix(const basic_matrix_2d<tgraphics_math_type>& m) noexcept;
```

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

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

13.3.6.6 Observers

[io2d.absmatrix.obs]

```
basic_matrix_2d<typename GraphicsSurfaces::graphics_math_type> matrix() const noexcept;
```

1 *Returns:* The transform matrix.

13.3.6.7 Equality operators

[io2d.absmatrix.eq]

```
template <class GraphicsSurfaces>
```

```
bool operator==(
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& lhs,
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& rhs)
```

```
noexcept;
```

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

```
template <class GraphicsSurfaces>
```

```
bool operator!=(
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& lhs,
```

```
    const typename basic_figure_items<GraphicsSurfaces>::abs_matrix& rhs)
```

```
noexcept;
```

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

13.3.7 Class template `basic_figure_items<GraphicsSurfaces>::rel_matrix`

[io2d.relmatrix]

13.3.7.1 Overview

[io2d.relmatrix.intro]

1 The class template `basic_figure_items<GraphicsSurfaces>::rel_matrix` describes a figure item that is a path command.

2 It has a transform matrix of type `basic_matrix_2d<GraphicsSurfaces::graphics_math_type>`.

3 The data are stored in an object of type `typename GraphicsSurfaces::paths::rel_matrix_data_type`. It is accessible using the data member functions.

13.3.7.2 Synopsis

[io2d.relmatrix.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::rel_matrix {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::rel_matrix_data_type;

        // 13.3.7.3, construct:
        rel_matrix();
        explicit rel_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
        rel_matrix(const rel_matrix& other) = default;
        rel_matrix(rel_matrix&& other) noexcept = default;

        // assign:
        rel_matrix& operator=(const rel_matrix& other) = default;
        rel_matrix& operator=(rel_matrix&& other) noexcept = default;

        // 13.3.7.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.7.5, modifiers:
        void matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;

        // 13.3.7.6, observers:
        basic_matrix_2d<graphics_math_type> matrix() const noexcept;
    };

    // 13.3.7.7, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& rhs)
        noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& rhs)
        noexcept;
}

```

13.3.7.3 Constructors

[io2d.relmatrix.ctor]

```
rel_matrix() noexcept;
```

1 *Effects:* Equivalent to: `rel_matrix{ basic_matrix_2d() }`;

2 *Postconditions:* `data() == GraphicsSurfaces::paths::create_rel_matrix()`.

```
explicit rel_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
```

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

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

5 *Remarks:* The transform matrix is `m`.

13.3.7.4 Accessors

[io2d.relmatrix.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `rel_matrix` object's data object (See: [13.3.7.1](#)).

13.3.7.5 Modifiers

[io2d.relmatrix.mod]

```
void matrix(const basic_matrix_2d<tgraphics_math_type>& m) noexcept;
```

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

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

13.3.7.6 Observers

[io2d.relmatrix.obs]

```
basic_matrix_2d<typename GraphicsSurfaces::graphics_math_type> matrix() const noexcept;
```

1 *Returns:* The transform matrix.

13.3.7.7 Equality operators

[io2d.relmatrix.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& rhs)
noexcept;
```

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

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_matrix& rhs)
noexcept;
```

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

13.3.8 Class template `basic_figure_items<GraphicsSurfaces>::revert_matrix`
[io2d.revertmatrix]**13.3.8.1 Overview**

[io2d.revertmatrix.intro]

1 The class template `basic_figure_items<GraphicsSurfaces>::revert_matrix` describes a figure item that is a path command.

13.3.8.2 Synopsis

[io2d.revertmatrix.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::revert_matrix {
    public:
        // construct:
        revert_matrix() = default;
        revert_matrix(const revert_matrix& other) = default;
        revert_matrix(revert_matrix&& other) noexcept = default;

        // assign:
        revert_matrix& operator=(const revert_matrix& other) = default;
        revert_matrix& operator=(revert_matrix&& other) noexcept = default;
    };

    // 13.3.8.3, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& rhs)
    noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& rhs)
    noexcept;
}
```

13.3.8.3 Equality operators

[io2d.revertmatrix.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& rhs)
noexcept;
```

1 *Returns: true.*

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::revert_matrix& rhs)
noexcept;
```

2 *Returns: false.*

13.3.9 Class template `basic_figure_items<GraphicsSurfaces>::abs_cubic_curve` [io2d.abscubiccurve]

13.3.9.1 Overview

[io2d.abscubiccurve.intro]

- 1 The class `basic_figure_items<GraphicsSurfaces>::abs_cubic_curve` describes a figure item that is a segment.
- 2 It has a *first control point* of type `basic_point_2d<GraphicsSurfaces::graphics_math_type>`, a *second control point* of type `basic_point_2d<GraphicsSurfaces::graphics_math_type>`, and an end point of type `basic_point_2d<GraphicsSurfaces::graphics_math_type>`.
- 3 The data are stored in an object of type `typename GraphicsSurfaces::paths::abs_cubic_curve_data_`-type. It is accessible using the data member functions.

13.3.9.2 Synopsis

[io2d.abscubiccurve.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::abs_cubic_curve {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::abs_cubic_curve_data_type;

        // 13.3.9.3, construct:
        abs_cubic_curve();
        abs_cubic_curve(const basic_point_2d<graphics_math_type>& cpt1,
            const basic_point_2d<graphics_math_type>& cpt2,
            const basic_point_2d<graphics_math_type>& ept) noexcept;
        abs_cubic_curve(const abs_cubic_curve& other) = default;
        abs_cubic_curve(abs_cubic_curve&& other) noexcept = default;

        // assign:
        abs_cubic_curve& operator=(const abs_cubic_curve& other) = default;
        abs_cubic_curve& operator=(abs_cubic_curve&& other) noexcept = default;

        // 13.3.9.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.9.5, modifiers:
        void control_pt1(const basic_point_2d<graphics_math_type>& cpt) noexcept;
        void control_pt2(const basic_point_2d<graphics_math_type>& cpt) noexcept;
        void end_pt(const basic_point_2d<graphics_math_type>& ept) noexcept;

        // 13.3.9.6, observers:
        basic_point_2d<graphics_math_type> control_pt1() const noexcept;
        basic_point_2d<graphics_math_type> control_pt2() const noexcept;
        basic_point_2d<graphics_math_type> end_pt() const noexcept;
    };
}
```



```

};

// 13.3.9.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& rhs)
    noexcept;
}

```

13.3.9.3 Constructors [io2d.abscubiccurve.ctor]

```
abs_cubic_curve() noexcept;
```

1 *Effects:* Equivalent to `abs_cubic_curve{ basic_point_2d(), basic_point_2d(), basic_point_2d() }`.

```
abs_cubic_curve(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& cpt1,
    const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& cpt2,
    const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& ept) noexcept;
```

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

3 *Remarks:* The first control point is `cpt1`.

4 *Remarks:* The second control point is `cpt2`.

5 *Remarks:* The end point is `ept`.

13.3.9.4 Accessors [io2d.abscubiccurve.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `rel_matrix` object's data object (See: [13.3.9.1](#)).

13.3.9.5 Modifiers [io2d.abscubiccurve.mod]

```
void control_pt1(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& cpt) noexcept;
```

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

```
void control_pt2(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& cpt) noexcept;
```

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

```
void end_pt(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& ept) noexcept;
```

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

13.3.9.6 Observers [io2d.abscubiccurve.obs]

```
basic_point_2d<graphics_math_type> control_pt1() const noexcept;
```

1 *Returns:* The first control point.

```
basic_point_2d<graphics_math_type> control_pt2() const noexcept;
```

2 *Returns:* The second control point.

```
basic_point_2d<graphics_math_type> end_pt() const noexcept;
```

3 *Returns:* The end point.

13.3.9.7 Equality operators

[io2d.abscubiccurve.eq]

```
template <class GraphicsSurfaces>
```

```
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& rhs)
    noexcept;
```

1 *Returns:* lhs.control_pt1() == rhs.control_pt1() && lhs.control_pt2() == rhs.control_pt2()
&& lhs.end_pt() == rhs.end_pt().

```
template <class GraphicsSurfaces>
```

```
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_cubic_curve& rhs)
    noexcept;
```

2 *Returns:* lhs.control_pt1() != rhs.control_pt1() || lhs.control_pt2() != rhs.control_pt2()
|| lhs.end_pt() != rhs.end_pt().

13.3.10 Class template basic_figure_items<GraphicsSurfaces>::rel_cubic_curve

[io2d.relcubiccurve]

13.3.10.1 Overview

[io2d.relcubiccurve.intro]

- 1 The class basic_figure_items<GraphicsSurfaces>::rel_cubic_curve describes a figure item that is a segment.
- 2 It has a *first control point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type>, a *second control point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type>, and an end point of type basic_point_2d<GraphicsSurfaces::graphics_math_type>.
- 3 The data are stored in an object of type typename GraphicsSurfaces::paths::rel_cubic_curve_data_-type. It is accessible using the data member functions.

13.3.10.2 Synopsis

[io2d.relcubiccurve.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::rel_cubic_curve {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::rel_cubic_curve_data_type;

        // 13.3.10.3, construct:
        rel_cubic_curve();
        rel_cubic_curve(const basic_point_2d<graphics_math_type>& cpt1,
            const basic_point_2d<graphics_math_type>& cpt2,
            const basic_point_2d<graphics_math_type>& ept) noexcept;
        rel_cubic_curve(const rel_cubic_curve& other) = default;
        rel_cubic_curve(rel_cubic_curve&& other) noexcept = default;

        // assign:
        rel_cubic_curve& operator=(const rel_cubic_curve& other) = default;
        rel_cubic_curve& operator=(rel_cubic_curve&& other) noexcept = default;

        // 13.3.10.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.10.5, modifiers:
        void control_pt1(const basic_point_2d<graphics_math_type>& cpt) noexcept;
        void control_pt2(const basic_point_2d<graphics_math_type>& cpt) noexcept;
        void end_pt(const basic_point_2d<graphics_math_type>& ept) noexcept;
    };
}
```

```

// 13.3.10.6, observers:
basic_point_2d<graphics_math_type> control_pt1() const noexcept;
basic_point_2d<graphics_math_type> control_pt2() const noexcept;
basic_point_2d<graphics_math_type> end_pt() const noexcept;
};

// 13.3.10.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& rhs)
    noexcept;
}

```

13.3.10.3 Constructors

[io2d.relcubiccurve.ctor]

```
rel_cubic_curve() noexcept;
```

1 *Effects:* Equivalent to `rel_cubic_curve{ basic_point_2d(), basic_point_2d(), basic_point_2d() }`.

```
rel_cubic_curve(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& cpt1,
    const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& cpt2,
    const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& ept) noexcept;
```

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

3 *Remarks:* The first control point is `cpt1`.

4 *Remarks:* The second control point is `cpt2`.

5 *Remarks:* The end point is `ept`.

13.3.10.4 Accessors

[io2d.relcubiccurve.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `rel_matrix` object's data object (See: [13.3.10.1](#)).

13.3.10.5 Modifiers

[io2d.relcubiccurve.mod]

```
void control_pt1(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& cpt) noexcept;
```

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

```
void control_pt2(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& cpt) noexcept;
```

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

```
void end_pt(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& ept) noexcept;
```

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

13.3.10.6 Observers

[io2d.relcubiccurve.obs]

```
basic_point_2d<graphics_math_type> control_pt1() const noexcept;
```

1 *Returns:* The first control point.

```
basic_point_2d<graphics_math_type> control_pt2() const noexcept;
```

2 *Returns:* The second control point.

```
basic_point_2d<graphics_math_type> end_pt() const noexcept;
```

3 *Returns:* The end point.

13.3.10.7 Equality operators

[io2d.relcubiccurve.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& rhs)
noexcept;
```

1 *Returns:* lhs.control_pt1() == rhs.control_pt1() && lhs.control_pt2() == rhs.control_pt2()
&& lhs.end_pt() == rhs.end_pt().

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_cubic_curve& rhs)
noexcept;
```

2 *Returns:* lhs.control_pt1() != rhs.control_pt1() || lhs.control_pt2() != rhs.control_pt2()
|| lhs.end_pt() != rhs.end_pt().

13.3.11 Class template basic_figure_items<GraphicsSurfaces>::abs_line

[io2d.absline]

13.3.11.1 Overview

[io2d.absline.intro]

- 1 The class basic_figure_items<GraphicsSurfaces>::abs_line describes a figure item that is a segment.
- 2 It has an *end point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type>.
- 3 The data are stored in an object of type typename GraphicsSurfaces::paths::abs_line_data_type. It is accessible using the data member functions.

13.3.11.2 Synopsis

[io2d.absline.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::abs_line {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::abs_line_data_type;

        // 13.3.11.3, construct:
        abs_line();
        explicit abs_line(const basic_point_2d<graphics_math_type>& pt);
        abs_line(const abs_line& other) = default;
        abs_line(abs_line&& other) noexcept = default;

        // assign:
        abs_line& operator=(const abs_line& other) = default;
        abs_line& operator=(abs_line&& other) noexcept = default;

        // 13.3.11.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.11.5, modifiers:
        void at(const basic_point_2d<graphics_math_type>& pt) noexcept;

        // 13.3.11.6, observers:
        basic_point_2d<graphics_math_type> at() const noexcept;
    };
};
```

```

// 13.3.11.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& rhs)
noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& rhs)
noexcept;
}

```

13.3.11.3 Constructors

[io2d.absline.ctor]

```
abs_line() noexcept;
```

1 *Effects:* Equivalent to: `abs_line{ basic_point_2d() }`;

```
explicit abs_line(const basic_point_2d<typename
GraphicsSurfaces::graphics_math_type>& pt) noexcept;
```

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

3 *Remarks:* The end point is `pt`.

13.3.11.4 Accessors

[io2d.absline.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `abs_line` object's data object (See: 13.3.11.1).

13.3.11.5 Modifiers

[io2d.absline.mod]

```
void to(const basic_point_2d<graphics_math_type>& pt) noexcept;
```

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

13.3.11.6 Observers

[io2d.absline.obs]

```
basic_point_2d<graphics_math_type> to() const noexcept;
```

1 *Returns:* The end point.

13.3.11.7 Equality operators

[io2d.absline.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& rhs)
noexcept;
```

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

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_line& rhs)
noexcept;
```

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

13.3.12 Class `rel_line`

[io2d.relline]

13.3.12.1 Overview

[io2d.relline.intro]

- 1 The class `basic_figure_items<GraphicsSurfaces>::rel_line` describes a figure item that is a segment.
- 2 It has an *end point* of type `basic_point_2d<GraphicsSurfaces::graphics_math_type>`.
- 3 The data are stored in an object of type `typename GraphicsSurfaces::paths::rel_line_data_type`. It is accessible using the data member functions.

13.3.12.2 Synopsis

[io2d.relline.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::rel_line {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::rel_line_data_type;

        // 13.3.12.3, construct:
        rel_line();
        explicit rel_line(const basic_point_2d<graphics_math_type>& pt);
        rel_line(const rel_line& other) = default;
        rel_line(rel_line&& other) noexcept = default;

        // assign:
        rel_line& operator=(const rel_line& other) = default;
        rel_line& operator=(rel_line&& other) noexcept = default;

        // 13.3.12.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.12.5, modifiers:
        void at(const basic_point_2d<graphics_math_type>& pt) noexcept;

        // 13.3.12.6, observers:
        basic_point_2d<graphics_math_type> at() const noexcept;
    };

    // 13.3.12.7, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::rel_line& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::rel_line& rhs)
        noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::rel_line& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::rel_line& rhs)
        noexcept;
}

```

13.3.12.3 Constructors

[io2d.relline.ctor]

```
rel_line() noexcept;
```

1 *Effects:* Equivalent to: `rel_line{ basic_point_2d() }`;

```
explicit rel_line(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt) noexcept;
```

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

3 *Remarks:* The end point is `pt`.

13.3.12.4 Accessors

[io2d.relline.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `rel_line` object's data object (See: [13.3.12.1](#)).

13.3.12.5 Modifiers

[io2d.relline.mod]

```
void to(const basic_point_2d<graphics_math_type>& pt) noexcept;
```

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

13.3.12.6 Observers

[io2d.relline.obs]

```
basic_point_2d<graphics_math_type> to() const noexcept;
```

1 *Returns:* The end point.

13.3.12.7 Equality operators

[io2d.relline.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& rhs)
noexcept;
```

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

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::rel_line& rhs)
noexcept;
```

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

13.3.13 Class template basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve
[io2d.absquadraticcurve]**13.3.13.1 Overview**

[io2d.absquadraticcurve.intro]

- 1 The class basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve describes a figure item that is a segment.
- 2 It has a *control point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type> and an *end point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type>.
- 3 The data are stored in an object of type typename GraphicsSurfaces::paths::abs_quadratic_curve_data_type. It is accessible using the data member functions.

13.3.13.2 Synopsis

[io2d.absquadraticcurve.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::abs_quadratic_curve_data_type;

        // 13.3.13.3, construct:
        abs_quadratic_curve();
        abs_quadratic_curve(const basic_point_2d<graphics_math_type>& cpt,
            const basic_point_2d<graphics_math_type>& ept);
        abs_quadratic_curve(const abs_quadratic_curve& other) = default;
        abs_quadratic_curve(abs_quadratic_curve&& other) noexcept = default;

        // assign:
        abs_quadratic_curve& operator=(const abs_quadratic_curve& other) = default;
        abs_quadratic_curve& operator=(abs_quadratic_curve&& other) noexcept = default;

        // 13.3.13.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.13.5, modifiers:
        void control_pt(const basic_point_2d<graphics_math_type>& cpt) noexcept;
        void end_pt(const basic_point_2d<graphics_math_type>& ept) noexcept;
    };
}
```

```

// 13.3.13.6, observers:
basic_point_2d<graphics_math_type> control_pt() const noexcept;
basic_point_2d<graphics_math_type> end_pt() const noexcept;
};

// 13.3.13.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& rhs)
    noexcept;
}

```

13.3.13.3 Constructors

[io2d.absquadraticcurve.ctor]

```
abs_quadratic_curve() noexcept;
```

1 *Effects:* Equivalent to: `abs_quadratic_curve{ basic_point_2d(), basic_point_2d() }`;

```
abs_quadratic_curve(const basic_point_2d<graphics_math_type>& cpt,
    const basic_point_2d<graphics_math_type>& ept) noexcept;
```

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

3 *Remarks:* The control point is `cpt`.

4 *Remarks:* The end point is `ept`.

13.3.13.4 Accessors

[io2d.absquadraticcurve.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `abs_quadratic_curve` object's data object (See: [13.3.13.1](#)).

13.3.13.5 Modifiers

[io2d.absquadraticcurve.mod]

```
void control_pt(const basic_point_2d<graphics_math_type>& cpt) noexcept;
```

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

```
void end_pt(const basic_point_2d<graphics_math_type>& ept) noexcept;
```

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

13.3.13.6 Observers

[io2d.absquadraticcurve.obs]

```
basic_point_2d<graphics_math_type> control_pt() const noexcept;
```

1 *Returns:* The control point.

```
basic_point_2d<graphics_math_type> end_pt() const noexcept;
```

2 *Returns:* The end point.

13.3.13.7 Equality operators

[io2d.absquadraticcurve.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& rhs)
    noexcept;
```

1 *Returns:* `lhs.control_pt() == rhs.control_pt() && lhs.end_pt() == rhs.end_pt()`.


```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::abs_quadratic_curve& rhs)
noexcept;
```

² *Returns:* lhs.control_pt() != rhs.control_pt() || lhs.end_pt() != rhs.end_pt().

13.3.14 Class template basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve [io2d.relquadraticcurve]

13.3.14.1 Overview [io2d.relquadraticcurve.intro]

- ¹ The class basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve describes a figure item that is a segment.
- ² It has a *control point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type> and an *end point* of type basic_point_2d<GraphicsSurfaces::graphics_math_type>.
- ³ The data are stored in an object of type typename GraphicsSurfaces::paths::rel_quadratic_curve_data_type. It is accessible using the data member functions.

13.3.14.2 Synopsis [io2d.relquadraticcurve.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::rel_quadratic_curve_data_type;

        // 13.3.14.3, construct:
        rel_quadratic_curve();
        rel_quadratic_curve(const basic_point_2d<graphics_math_type>& cpt,
            const basic_point_2d<graphics_math_type>& ept);
        rel_quadratic_curve(const rel_quadratic_curve& other) = default;
        rel_quadratic_curve(rel_quadratic_curve&& other) noexcept = default;

        // assign:
        rel_quadratic_curve& operator=(const rel_quadratic_curve& other) = default;
        rel_quadratic_curve& operator=(rel_quadratic_curve&& other) noexcept = default;

        // 13.3.14.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.14.5, modifiers:
        void control_pt(const basic_point_2d<graphics_math_type>& cpt) noexcept;
        void end_pt(const basic_point_2d<graphics_math_type>& ept) noexcept;

        // 13.3.14.6, observers:
        basic_point_2d<graphics_math_type> control_pt() const noexcept;
        basic_point_2d<graphics_math_type> end_pt() const noexcept;
    };

    // 13.3.14.7, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& rhs)
    noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& rhs)
    noexcept;
```

}

13.3.14.3 Constructors [io2d.relquadraticcurve.ctor]

rel_quadratic_curve() noexcept;

1 *Effects:* Equivalent to: rel_quadratic_curve{ basic_point_2d(), basic_point_2d() };rel_quadratic_curve(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& cpt,
const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& ept) noexcept;2 *Effects:* Constructs an object of type rel_quadratic_curve.3 *Remarks:* The control point is cpt.4 *Remarks:* The end point is ept.**13.3.14.4 Accessors** [io2d.relquadraticcurve.acc]

const data_type& data() const noexcept;

data_type& data() noexcept;

1 *Returns:* A reference to the rel_quadratic_curve object's data object (See: 13.3.14.1).**13.3.14.5 Modifiers** [io2d.relquadraticcurve.mod]

void control_pt(const basic_point_2d<graphics_math_type>& cpt) noexcept;

1 *Effects:* The control point is cpt.

void end_pt(const basic_point_2d<graphics_math_type>& ept) noexcept;

2 *Effects:* The end point is ept.**13.3.14.6 Observers** [io2d.relquadraticcurve.obs]

basic_point_2d<graphics_math_type> control_pt() const noexcept;

1 *Returns:* The control point.

basic_point_2d<graphics_math_type> end_pt() const noexcept;

2 *Returns:* The end point.**13.3.14.7 Equality operators** [io2d.relquadraticcurve.eq]template <class GraphicsSurfaces>
bool operator==(
const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& lhs,
const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& rhs)
noexcept;1 *Returns:* lhs.control_pt() == rhs.control_pt() && lhs.end_pt() == rhs.end_pt().template <class GraphicsSurfaces>
bool operator!=(
const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& lhs,
const typename basic_figure_items<GraphicsSurfaces>::rel_quadratic_curve& rhs)
noexcept;2 *Returns:* lhs.control_pt() != rhs.control_pt() || lhs.end_pt() != rhs.end_pt().**13.3.15 Class template basic_figure_items<GraphicsSurfaces>::arc** [io2d.arc]**13.3.15.1 Overview** [io2d.arc.intro]

- 1 The class basic_figure_items<GraphicsSurfaces>::arc describes a figure item that is a segment.
- 2 It has a *radius* of type basic_point_2d<GraphicsSurfaces::graphics_math_type>, a *rotation* of type float, and a *start angle* of type float.
- 3 It forms a portion of the circumference of a circle. The centre of the circle is implied by the start point, the radius and the start angle of the arc.

- ⁴ The data are stored in an object of type `typename GraphicsSurfaces::paths::arc_data_type`. It is accessible using the data member functions.

13.3.15.2 Synopsis

[io2d.arc.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_figure_items<GraphicsSurfaces>::arc {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::paths::arc_data_type;

        // 13.3.15.3, construct:
        arc();
        arc(const basic_point_2d<graphics_math_type>& rad, float rot, float sang) noexcept;
        arc(const arc& other) = default;
        arc(arc&& other) noexcept = default;

        // assign:
        arc& operator=(const arc& other) = default;
        arc& operator=(arc&& other) noexcept = default;

        // 13.3.15.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 13.3.15.5, modifiers:
        void radius(const basic_point_2d<graphics_math_type>& rad) noexcept;
        void rotation(float rot) noexcept;
        void start_angle(float radians) noexcept;

        // 13.3.15.6, observers:
        basic_point_2d<typename GraphicsSurfaces::graphics_math_type> radius() const noexcept;
        float rotation() const noexcept;
        float start_angle() const noexcept;
        basic_point_2d<graphics_math_type> center(const basic_point_2d< graphics_math_type>& cpt,
            const basic_matrix_2d<graphics_math_type>& m =
                basic_matrix_2d<graphics_math_type>{}) const noexcept;
        basic_point_2d<graphics_math_type> end_pt(const basic_point_2d<graphics_math_type>& cpt,
            const basic_matrix_2d<graphics_math_type>& m =
                basic_matrix_2d<graphics_math_type>{}) const noexcept;
    };

    // 13.3.15.7, equality operators:
    template <class GraphicsSurfaces>
    bool operator==(
        const typename basic_figure_items<GraphicsSurfaces>::arc& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::arc& rhs)
        noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(
        const typename basic_figure_items<GraphicsSurfaces>::arc& lhs,
        const typename basic_figure_items<GraphicsSurfaces>::arc& rhs)
        noexcept;
}
```

13.3.15.3 Constructors

[io2d.arc.ctor]

```
arc() noexcept;
```

- ¹ *Effects:* Equivalent to: `arc{ basic_point_2d(10.0f, 10.0f), pi<float>, pi<float> };`

```
arc(const basic_point_2d<graphics_math_type>& rad,
     float rot, float sang) noexcept;
```

- 2 *Effects:* Constructs an object of type `arc`.
 3 The radius is `rad`.
 4 The rotation is `rot`.
 5 The start angle is `sang`.

13.3.15.4 Accessors

[io2d.arc.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

- 1 *Returns:* A reference to the `arc` object's data object (See: [13.3.15.1](#)).

13.3.15.5 Modifiers

[io2d.arc.mod]

```
void radius(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& rad) noexcept;
```

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

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

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

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

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

13.3.15.6 Observers

[io2d.arc.obs]

```
basic_point_2d<typename GraphicsSurfaces::graphics_math_type> radius() const noexcept;
```

- 1 *Returns:* The radius.

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

- 2 *Returns:* The rotation.

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

- 3 *Returns:* The start angle.

```
basic_point_2d<graphics_math_type> center(
    const basic_point_2d<graphics_math_type>& cpt,
    const basic_matrix_2d<graphics_math_type>& m =
        basic_matrix_2d<graphics_math_type>{}) const noexcept;
```

- 4 *Returns:* As-if:

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

```
basic_point_2d<graphics_math_type> end_pt(
    const basic_point_2d<graphics_math_type>& cpt,
    const basic_matrix_2d<graphics_math_type>& m =
        basic_matrix_2d<graphics_math_type>{}) const noexcept;
```

- 5 *Returns:* As-if:

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

13.3.15.7 Equality operators

[io2d.arc.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_figure_items<GraphicsSurfaces>::arc& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::arc& rhs)
noexcept;
```

1 *Returns:*

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

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_figure_items<GraphicsSurfaces>::arc& lhs,
    const typename basic_figure_items<GraphicsSurfaces>::arc& rhs)
noexcept;
```

2 *Returns:*

```
lhs.radius() != rhs.radius() || lhs.rotation() != rhs.rotation() ||
lhs.start_angle() != rhs.start_angle()
```

13.3.16 Path interpretation

[io2d.paths.interpretation]

- 1 This subclause describes how to interpret a path for use in a rendering and composing operation.
- 2 Interpreting a path consists of sequentially evaluating the figure items contained in the figures in the path and transforming them into zero or more figures as-if in the manner specified in this subclause.
- 3 The interpretation of a path requires the state data specified in Table 25.

Table 25 — Path interpretation state data

Name	Description	Type	Initial value
mtx	Path transformation matrix	matrix_2d	matrix_2d{ }
currPt	Current point	point_2d	<i>unspecified</i>
lnfPt	Last new figure point	point_2d	<i>unspecified</i>
mtxStk	Matrix stack	stack<matrix_2d>	stack<matrix_2d>{ }

- 4 When interpreting a path, until a `figure_items::abs_new_figure` figure item is reached, a path shall only contain path command figure items; no diagnostic is required. If a figure is a degenerate figure, none of its figure items have any effects, with two exceptions:
 - (4.1) — the path's `figure_items::abs_new_figure` or `figure_items::rel_new_figure` figure item sets the value of `currPt` as-if the figure item was interpreted; and,
 - (4.2) — any path command figure items are evaluated with full effect.
- 5 The effects of a figure item contained in a `figure_items::figure_item` object when that object is being evaluated during path interpretation are described in Table 26.
- 6 If evaluation of a figure item contained in a `figure_items::figure_item` during path interpretation results in the figure item becoming a degenerate segment, its effects are ignored and interpretation continues as-if that figure item did not exist.

Table 26 — Figure item interpretation effects

Figure item	Effects
<code>figure_items::abs_new_figure p</code>	Creates a new figure. Sets <code>currPt</code> to <code>p.at() * mtx</code> . Sets <code>lnfPt</code> to <code>currPt</code> .
<code>figure_items::rel_new_figure p</code>	Let <code>mm</code> equal <code>mtx</code> . Let <code>mm.m20</code> equal <code>0.0f</code> . Let <code>mm.m21</code> equal <code>0.0f</code> . Creates a new figure. Sets <code>currPt</code> to <code>currPt + p.at() * mm</code> . Sets <code>lnfPt</code> to <code>currPt</code> .

Table 26 — Figure item interpretation effects (continued)

Figure item	Effects
<code>figure_items::close_figure p</code>	Creates a line from <code>currPt</code> to <code>lnfPt</code> . Makes the current figure a closed figure. Creates a new figure. Sets <code>currPt</code> to <code>lnfPt</code> .
<code>figure_items::abs_matrix p</code>	Calls <code>mtxStk.push(mtx)</code> . Sets <code>mtx</code> to <code>p.matrix()</code> .
<code>figure_items::rel_matrix p</code>	Calls <code>mtxStk.push(mtx)</code> . Sets <code>mtx</code> to <code>p.matrix() * mtx</code> .
<code>figure_items::revert_matrix p</code>	If <code>mtxStk.empty()</code> is false, sets <code>mtx</code> to <code>mtxStk.top()</code> then calls <code>mtxStk.pop()</code> . Otherwise sets <code>mtx</code> to its initial value as specified in Table 25.
<code>figure_items::abs_line p</code>	Let <code>pt</code> equal <code>p.to() * mtx</code> . Creates a line from <code>currPt</code> to <code>pt</code> . Sets <code>currPt</code> to <code>pt</code> .
<code>figure_items::rel_line p</code>	Let <code>mm</code> equal <code>mtx</code> . Let <code>mm.m20</code> equal 0.0f. Let <code>mm.m21</code> equal 0.0f. Let <code>pt</code> equal <code>currPt + p.to() * mm</code> . Creates a line from <code>currPt</code> to <code>pt</code> . Sets <code>currPt</code> to <code>pt</code> .
<code>figure_items::abs_quadratic_curve p</code>	Let <code>cpt</code> equal <code>p.control_pt() * mtx</code> . Let <code>ept</code> equal <code>p.end_pt() * mtx</code> . Creates a quadratic Bézier curve from <code>currPt</code> to <code>ept</code> using <code>cpt</code> as the curve's control point. Sets <code>currPt</code> to <code>ept</code> .
<code>figure_items::rel_quadratic_curve p</code>	Let <code>mm</code> equal <code>mtx</code> . Let <code>mm.m20</code> equal 0.0f. Let <code>mm.m21</code> equal 0.0f. Let <code>cpt</code> equal <code>currPt + p.control_pt() * mm</code> . Let <code>ept</code> equal <code>currPt + p.control_pt() * mm + p.end_pt() * mm</code> . Creates a quadratic Bézier curve from <code>currPt</code> to <code>ept</code> using <code>cpt</code> as the curve's control point. Sets <code>currPt</code> to <code>ept</code> .
<code>figure_items::abs_cubic_curve p</code>	Let <code>cpt1</code> equal <code>p.control_pt1() * mtx</code> . Let <code>cpt2</code> equal <code>p.control_pt2() * mtx</code> . Let <code>ept</code> equal <code>p.end_pt() * mtx</code> . Creates a cubic Bézier curve from <code>currPt</code> to <code>ept</code> using <code>cpt1</code> as the curve's first control point and <code>cpt2</code> as the curve's second control point. Sets <code>currPt</code> to <code>ept</code> .
<code>figure_items::rel_cubic_curve p</code>	Let <code>mm</code> equal <code>mtx</code> . Let <code>mm.m20</code> equal 0.0f. Let <code>mm.m21</code> equal 0.0f. Let <code>cpt1</code> equal <code>currPt + p.control_pt1() * mm</code> . Let <code>cpt2</code> equal <code>currPt + p.control_pt1() * mm + p.control_pt2() * mm</code> . Let <code>ept</code> equal <code>currPt + p.control_pt1() * mm + p.control_pt2() * mm + p.end_pt() * mm</code> . Creates a cubic Bézier curve from <code>currPt</code> to <code>ept</code> using <code>cpt1</code> as the curve's first control point and <code>cpt2</code> as the curve's second control point. Sets <code>currPt</code> to <code>ept</code> .
<code>figure_items::arc p</code>	Let <code>mm</code> equal <code>mtx</code> . Let <code>mm.m20</code> equal 0.0f. Let <code>mm.m21</code> equal 0.0f. Creates an arc. It begins at <code>currPt</code> , which is at <code>p.start_angle()</code> radians on the arc and rotates <code>p.rotation()</code> radians. If <code>p.rotation()</code> is positive, rotation is counterclockwise, otherwise it is clockwise. The center of the arc is located at <code>p.center(currPt, mm)</code> . The arc ends at <code>p.end_pt(currPt, mm)</code> . Sets <code>currPt</code> to <code>p.end_pt(currPt, mm)</code> . [<i>Note:</i> <code>p.radius()</code> , which specifies the radius of the arc, is implicitly included in the above statement of effects by the specifications of the center of the arc and the end of the arcs. The use of the current point as the origin for the application of the path transformation matrix is also implicitly included by the same specifications. — <i>end note</i>]

13.4 Class template `basic_interpreted_path`

[io2d.pathgroup]

13.4.1 Overview

[io2d.pathgroup.intro]

- ¹ The class template `basic_interpreted_path` contains the data that result from interpreting (13.3.16) a sequence of `basic_figure_items<GraphicsSurfaces>::figure_item` objects.
- ² The data are stored in an object of type `typename GraphicsSurfaces::paths::interpreted_path_data_type`. It is accessible using the `data` member function.
- ³ A `basic_interpreted_path` object is used by most rendering and composing operations.

13.4.2 basic_interpreted_path synopsis

[io2d.pathgroup.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_interpreted_path {
    public:
        using data_type = typename
            GraphicsSurfaces::paths::interpreted_path_data_type;

        // 13.4.3, construct:
        basic_interpreted_path() noexcept;
        explicit basic_interpreted_path(
            const basic_bounding_box<GraphicsMath>& bb);
        template <class Allocator>
        explicit basic_interpreted_path(
            const basic_path_builder<GraphicsSurfaces, Allocator>& pb);
        template <class InputIterator>
        basic_interpreted_path(InputIterator first, InputIterator last);
        explicit basic_interpreted_path(initializer_list<typename
            basic_figure_items<GraphicsSurfaces>::figure_item>> il);

        // 13.4.4, accessors:
        const data_type& data() const noexcept;
    };
}

```

13.4.3 basic_interpreted_path constructors

[io2d.pathgroup.ctor]

```
basic_interpreted_path() noexcept;
```

1 *Effects:* Constructs a `basic_interpreted_path` that contains an empty path.

```
explicit basic_interpreted_path(const basic_bounding_box<GraphicsMath>& bb);
```

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

3 *Postconditions:* `data() == GraphicsSurfaces::paths::create_interpreted_path(bb)`.

```
template <class Allocator>
explicit basic_interpreted_path(const basic_path_builder<GraphicsSurfaces, Allocator>& pb);
```

4 *Effects:* Equivalent to: `basic_interpreted_path{ begin(pb), end(pb) }`.

```
template <class InputIterator>
basic_interpreted_path(InputIterator first, InputIterator last);
```

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

6 *Postconditions:* `data() == GraphicsSurfaces::paths::create_interpreted_path(first, last)`.

7 [*Note:* The contained path is as-if it was the result of interpreting a path containing the values of the elements from `first` to the last element before `last`. — *end note*]

```
explicit basic_interpreted_path(initializer_list<typename
    basic_figure_items<GraphicsSurfaces>::figure_item> il);
```

8 *Effects:* Equivalent to `basic_interpreted_path{ il.begin(), il.end() }`.

13.4.4 Accessors

[io2d.pathgroup.acc]

```
const data_type& data() const noexcept;
```

1 *Returns:* A reference to the `basic_interpreted_path` object's data object (See: 13.4.1).

13.5 Class basic_path_builder

[io2d.pathbuilder]

1 The class `basic_path_builder` is a container that stores and manipulates objects of type `figure_items::figure_item` from which `interpreted_path` objects are created.

2 A `basic_path_builder` is a contiguous container. (See [container.requirements.general] in C++ 2017.)

3 The collection of `figure_items::figure_item` objects in a path builder is referred to as its path.

13.5.1 basic_path_builder synopsis

[io2d.pathbuilder.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces,
              class Allocator = allocator<typename
                basic_figure_items<GraphicsSurfaces>::figure_item>>
    class basic_path_builder {
    public:
        using value_type           = typename basic_figure_items<GraphicsSurfaces>::figure_item;
        using allocator_type       = Allocator;
        using reference            = value_type&;
        using const_reference      = const value_type&;
        using size_type            = implementation-defined. // See [container.requirements] in C++ 2017.
        using difference_type      = implementation-defined. // See [container.requirements] in C++ 2017.
        using iterator             = implementation-defined. // See [container.requirements] in C++ 2017.
        using const_iterator       = implementation-defined. // See [container.requirements] in C++ 2017.
        using reverse_iterator     = std::reverse_iterator<iterator>;
        using const_reverse_iterator = std::reverse_iterator<const_iterator>;

        // 13.5.3, construct, copy, move, destroy:
        basic_path_builder() noexcept(noexcept(Allocator()));
        explicit basic_path_builder(const Allocator&) noexcept;
        explicit basic_path_builder(size_type n, const Allocator& = Allocator());
        basic_path_builder(size_type n, const value_type& value, const Allocator& = Allocator());
        template <class InputIterator>
        basic_path_builder(InputIterator first, InputIterator last, const Allocator& = Allocator());
        basic_path_builder(const basic_path_builder& x);
        basic_path_builder(basic_path_builder&&) noexcept;
        basic_path_builder(const basic_path_builder&, const Allocator&);
        basic_path_builder(basic_path_builder&&, const Allocator&);
        basic_path_builder(initializer_list<value_type>, const Allocator& = Allocator());
        ~basic_path_builder();
        basic_path_builder& operator=(const basic_path_builder& x);
        basic_path_builder& operator=(basic_path_builder&& x) noexcept(
            allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
            allocator_traits<Allocator>::is_always_equal::value);
        basic_path_builder& operator=(initializer_list<value_type>);
        template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
        void assign(size_type n, const value_type& u);
        void assign(initializer_list<value_type>);
        allocator_type get_allocator() const noexcept;

        // 13.5.4, capacity
        bool empty() const noexcept;
        size_type size() const noexcept;
        size_type max_size() const noexcept;
        size_type capacity() const noexcept;
        void resize(size_type sz);
        void resize(size_type sz, const value_type& c);
        void reserve(size_type n);
        void shrink_to_fit();

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

```



```

// 13.5.5, modifiers:
void new_figure(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt) noexcept;
void rel_new_figure(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt) noexcept;
void close_figure() noexcept;
void matrix(const basic_matrix_2d<typename
    GraphicsSurfaces::graphics_math_type>& m) noexcept;
void rel_matrix(const basic_matrix_2d<typename
    GraphicsSurfaces::graphics_math_type>& m) noexcept;
void revert_matrix() noexcept;
void line(const basic_point_2d<typename GraphicsSurfaces::graphics_math_type>& pt) noexcept;
void rel_line(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& dpt) noexcept;
void quadratic_curve(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt0, const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt2) noexcept;
void rel_quadratic_curve(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt0, const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt2) noexcept;
void cubic_curve(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt0, const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt1, const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& pt2) noexcept;
void rel_cubic_curve(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& dpt0, const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& dpt1, const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& dpt2) noexcept;
void arc(const basic_point_2d<typename
    GraphicsSurfaces::graphics_math_type>& rad, float rot, float sang = pi<float>) noexcept;
template <class... Args>
reference emplace_back(Args&&... args);
void push_back(const value_type& x);
void push_back(value_type&& x);
void pop_back();
template <class... Args>
iterator emplace(const_iterator position, Args&&... args);
iterator insert(const_iterator position, const value_type& x);
iterator insert(const_iterator position, value_type&& x);
iterator insert(const_iterator position, size_type n, const value_type& x);
template <class InputIterator>
iterator insert(const_iterator position, InputIterator first, InputIterator last);
iterator insert(const_iterator position,
    initializer_list<value_type> il);
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
void swap(basic_path_builder&) noexcept(
    allocator_traits<Allocator>::propagate_on_container_swap::value ||
    allocator_traits<Allocator>::is_always_equal::value);
void clear() noexcept;

// 13.5.6, iterators:
iterator begin() noexcept;
const_iterator begin() const noexcept;
const_iterator cbegin() const noexcept;
iterator end() noexcept;
const_iterator end() const noexcept;
const_iterator cend() const noexcept;
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
const_reverse_iterator crbegin() const noexcept;
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
const_reverse_iterator crend() const noexcept;

```

```

};

template <class GraphicsSurfaces, class Allocator>
bool operator==(const basic_path_builder<GraphicsSurfaces, Allocator>& lhs,
    const basic_path_builder<GraphicsSurfaces, Allocator>& rhs) noexcept;
template <class GraphicsSurfaces, class Allocator>
bool operator!=(const basic_path_builder<GraphicsSurfaces, Allocator>& lhs,
    const basic_path_builder<GraphicsSurfaces, Allocator>& rhs) noexcept;
template <class GraphicsSurfaces, class Allocator>
void swap(basic_path_builder<GraphicsSurfaces, Allocator>& lhs,
    basic_path_builder<GraphicsSurfaces, Allocator>& rhs) noexcept(noexcept(lhs.swap(rhs)));
}

```

13.5.2 basic_path_builder container requirements [io2d.pathbuilder.containerrequirements]

- 1 This class is a sequence container, as defined in [containers] in C++ 2017, and all sequence container requirements that apply specifically to `vector` shall also apply to this class.

13.5.3 basic_path_builder constructors, copy, and assignment [io2d.pathbuilder.cons]

```
basic_path_builder() noexcept(noexcept(Allocator()));
```

- 1 *Effects:* Constructs an empty `basic_path_builder`.

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

- 2 *Effects:* Constructs an empty `basic_path_builder`, using the specified allocator.

- 3 *Complexity:* Constant.

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

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

- 5 *Complexity:* Linear in `n`.

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

- 6 *Requires:* `value_type` shall be `CopyInsertable` into `*this`.

- 7 *Effects:* Constructs a `basic_path_builder` with `n` copies of `value`, using the specified allocator.

- 8 *Complexity:* Linear in `n`.

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

- 9 *Effects:* Constructs a `basic_path_builder` equal to the range `[first, last)`, using the specified allocator.

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

13.5.4 basic_path_builder capacity [io2d.pathbuilder.capacity]

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

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

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

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

- 3 *Effects:* A directive that informs a path builder of a planned change in size, so that it can manage the storage allocation accordingly. After `reserve()`, `capacity()` is greater or equal to the argument of `reserve` if reallocation happens; and equal to the previous value of `capacity()` otherwise. Reallocation happens at this point if and only if the current capacity is less than the argument of `reserve()`. If an

exception is thrown other than by the move constructor of a non-CopyInsertable type, there are no effects.

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

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

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

```
void shrink_to_fit();
```

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

8 *Effects:* `shrink_to_fit` is a non-binding request to reduce `capacity()` to `size()`. [*Note:* The request is non-binding to allow latitude for implementation-specific optimizations. — *end note*] It does not increase `capacity()`, but may reduce `capacity()` by causing reallocation. If an exception is thrown other than by the move constructor of a non-CopyInsertable `value_type` there are no effects.

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

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

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

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

12 *Complexity:* Constant time.

resize

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

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

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

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

resize

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

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

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

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

13.5.5 basic_path_builder modifiers

[io2d.pathbuilder.modifiers]

```
void new_figure(point_2d pt) noexcept;
```

1 *Effects:* Adds a `figure_items::figure_item` object constructed from `figure_items::abs_new_figure(pt)` to the end of the path.

```
void rel_new_figure(point_2d pt) noexcept;
```

2 *Effects:* Adds a `figure_items::figure_item` object constructed from `figure_items::rel_new_figure(pt)` to the end of the path.

```
void close_figure() noexcept;
```

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

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

4 *Effects:* Adds a `figure_items::figure_item` object constructed from `figure_items::close_figure()` to the end of the path.

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

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

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

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

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

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

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

9 *Effects:* Adds a `figure_items::figure_item` object constructed from `(figure_items::revert_matrix())` to the end of the path.

```
void line(point_2d pt) noexcept;
```

10 Adds a `figure_items::figure_item` object constructed from `figure_items::abs_line(pt)` to the end of the path.

```
void rel_line(point_2d dpt) noexcept;
```

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

```
void quadratic_curve(point_2d pt0, point_2d pt1) noexcept;
```

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

```
void rel_quadratic_curve(point_2d dpt0, point_2d dpt1)
noexcept;
```

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

```
void cubic_curve(point_2d pt0, point_2d pt1,
point_2d pt2) noexcept;
```

14 ¹*Effects:* Adds a `figure_items::figure_item` object constructed from `figure_items::abs_cubic_curve(pt0, pt1, pt2)` to the end of the path.

```
void rel_cubic_curve(point_2d dpt0, point_2d dpt1,
point_2d dpt2) noexcept;
```

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

```
void arc(point_2d rad, float rot, float sang) noexcept;
```

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

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

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

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

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

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

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

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

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

13.5.6 basic_path_builder iterators

[io2d.pathbuilder.iterators]

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

1 *Returns:* An iterator referring to the first `figure_items::figure_item` item in the path.

2 *Remarks:* Changing a `figure_items::figure_item` object or otherwise modifying the path in a way that violates the preconditions of that `figure_items::figure_item` object or of any subsequent `figure_items::figure_item` object in the path produces undefined behavior when the path is interpreted as described in 13.3.16 unless all of the violations are fixed prior to such interpretation.

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

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

4 *Remarks:* Changing a `figure_items::figure_item` object or otherwise modifying the path in a way that violates the preconditions of that `figure_items::figure_item` object or of any subsequent `figure_items::figure_item` object in the path produces undefined behavior when the path is interpreted as described in 13.3.16 unless all of the violations are fixed prior to such interpretation.

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

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

6 *Remarks:* Changing a `figure_items::figure_item` object or otherwise modifying the path in a way that violates the preconditions of that `figure_items::figure_item` object or of any subsequent `figure_items::figure_item` object in the path produces undefined behavior when the path is interpreted as described in 13.3.16 all of the violations are fixed prior to such interpretation.

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

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

- ⁸ *Remarks:* Changing a `figure_items::figure_item` object or otherwise modifying the path in a way that violates the preconditions of that `figure_items::figure_item` object or of any subsequent `figure_items::figure_item` object in the path produces undefined behavior when the path is interpreted as described in 13.3.16 unless all of the violations are fixed prior to such interpretation.

13.5.7 `basic_path_builder` specialized algorithms [io2d.pathbuilder.special]

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

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

14 Brushes

[io2d.brushes]

14.1 Overview of brushes

[io2d.brushes.general]

- 1 Brushes contain visual data and serve as sources of visual data for rendering and composing operations.
- 2 There are four types of brushes:
 - (2.1) — solid color;
 - (2.2) — linear gradient;
 - (2.3) — radial gradient; and,
 - (2.4) — surface.
- 3 Once a brush is created, its visual data is immutable.
- 4 [*Note*: While copy and move operations along with a swap operation can change the visual data that a brush contains, the visual data itself is not modified. — *end note*]
- 5 A brush is used either as a *source brush* or a *mask brush* (16.3.2.2).
- 6 When a brush is used in a rendering and composing operation, if it is used as a source brush, it has a **brush_props** object that describes how the brush is interpreted for purposes of sampling. If it is used as a mask brush, it has a **mask_props** object that describes how the brush is interpreted for purposes of sampling.
- 7 The **basic_brush_props** (15.10.1) and **basic_mask_props** (15.14.1) class templates each have a *wrap mode* and a *filter*. The **basic_brush_props** class template also has a *brush matrix* and a *fill rule*. The **basic_mask_props** class template also has a *mask matrix*. Where possible, the terms that are common between the two classes are referenced without regard to whether the brush is being used as a source brush or a mask brush.
- 8 Solid color brushes are unbounded and as such always produce the same visual data when sampled from, regardless of the requested point.
- 9 Linear gradient and radial gradient brushes share similarities with each other that are not shared by the other types of brushes. This is discussed in more detail elsewhere (14.2).
- 10 Surface brushes are constructed from a **basic_image_surface** object. Their visual data is raster graphics data, which has implications on sampling from the brush that are not present in the other brush types.

14.2 Gradient brushes

[io2d.gradients]

14.2.1 Common properties of gradients

[io2d.gradients.common]

- 1 Gradients are formed, in part, from a collection of **gradient_stop** objects.
- 2 The collection of **gradient_stop** objects contribute to defining a brush which, when sampled from, returns a value that is interpolated based on those gradient stops.

14.2.2 Linear gradients

[io2d.gradients.linear]

- 1 A linear gradient is a type of gradient.
- 2 A linear gradient has a *begin point* and an *end point*, each of which are objects of type **basic_point_2d<GraphicsMath>**.
- 3 A linear gradient for which the distance between its begin point and its end point is 0 is a *degenerate linear gradient*.
- 4 All attempts to sample from a a degenerate linear gradient return the color **rgba_color::transparent_black**. The remainder of 14.2 is inapplicable to degenerate linear gradients. [*Note*: Because a point has no width and this case is only met when the distance is between the begin point and the end point is zero (such that it collapses to a single point), the existence of one or more gradient stops is irrelevant. A linear gradient requires a line segment to define its color(s). Without a line segment, it is not a linear gradient. — *end note*]
- 5 The begin point and end point of a linear gradient define a line segment, with a gradient stop offset value of 0.0f corresponding to the begin point and a gradient stop offset value of 1.0f corresponding to the end point.

- 6 Gradient stop offset values in the range `[0.0f, 1.0f]` linearly correspond to points on the line segment.
- 7 [Example: Given a linear gradient with a begin point of `basic_point_2d<GraphicsMath>(0.0f, 0.0f)` and an end point of `basic_point_2d<GraphicsMath>(10.0f, 5.0f)`, a gradient stop offset value of `0.6f` would correspond to the point `basic_point_2d<GraphicsMath>(6.0f, 3.0f)`. — end example]
- 8 To determine the offset value of a point p for a linear gradient, perform the following steps:
- a) Create a line at the begin point of the linear gradient, the *begin line*, and another line at the end point of the linear gradient, the *end line*, with each line being perpendicular to the *gradient line segment*, which is the line segment delineated by the begin point and the end point.
 - b) Using the begin line, p , and the end line, create a line, the *p line*, which is parallel to the gradient line segment.
 - c) Defining dp as the distance between p and the point where the *p line* intersects the begin line and dt as the distance between the point where the *p line* intersects the begin line and the point where the *p line* intersects the end line, the offset value of p is $dp \div dt$.
 - d) The offset value shall be negative if
 - (8.1) — p is not on the line segment delineated by the point where the *p line* intersects the begin line and the point where the *p line* intersects the end line; and,
 - (8.2) — the distance between p and the point where the *p line* intersects the begin line is less than the distance between p and the point where the *p line* intersects the end line.

14.2.3 Radial gradients

[io2d.gradients.radial]

- 1 A radial gradient is a type of gradient.
- 2 A radial gradient has a *start circle* and an *end circle*, each of which is defined by a `basic_circle<GraphicsMath>` object.
- 3 A radial gradient is a *degenerate radial gradient* if:
- (3.1) — its start circle has a negative radius; or,
 - (3.2) — its end circle has a negative radius; or,
 - (3.3) — the distance between the center point of its start circle and the center point of its end circle is 0; or,
 - (3.4) — its start circle has a radius of `0.0f` and its end circle has a radius of `0.0f`.
- 4 All attempts to sample from a `brush` object created using a degenerate radial gradient return the color `rgba_color::transparent_black`. The remainder of 14.2 is inapplicable to degenerate radial gradients.
- 5 A gradient stop offset of `0.0f` corresponds to all points along the diameter of the start circle or to its center point if it has a radius value of `0.0f`.
- 6 A gradient stop offset of `1.0f` corresponds to all points along the diameter of the end circle or to its center point if it has a radius value of `0.0f`.
- 7 A radial gradient shall be rendered as a continuous series of interpolated circles defined by the following equations:
- a) $x(o) = x_{start} + o \times (x_{end} - x_{start})$
 - b) $y(o) = y_{start} + o \times (y_{end} - y_{start})$
 - c) $radius(o) = radius_{start} + o \times (radius_{end} - radius_{start})$
- where o is a gradient stop offset value.
- 8 The range of potential values for o shall be determined by the *wrap mode* (14.1):
- (8.1) — For `wrap_mode::none`, the range of potential values for o is `[0, 1]`.
 - (8.2) — For all other `wrap_mode` values, the range of potential values for o is `[numeric_limits<float>::lowest(), numeric_limits<float>::max()]`.
- 9 The interpolated circles shall be rendered starting from the smallest potential value of o .
- 10 An interpolated circle shall not be rendered if its value for o results in $radius(o)$ evaluating to a negative value.

14.2.4 Sampling from gradients

[io2d.gradients.sampling]

¹ For any offset value o , its color value shall be determined according to the following rules:

- a) If there are less than two gradient stops or if all gradient stops have the same offset value, then the color value of every offset value shall be `rgba_color::transparent_black` and the remainder of these rules are inapplicable.
- b) If exactly one gradient stop has an offset value equal to o , o 's color value shall be the color value of that gradient stop and the remainder of these rules are inapplicable.
- c) If two or more gradient stops have an offset value equal to o , o 's color value shall be the color value of the gradient stop which has the lowest index value among the set of gradient stops that have an offset value equal to o and the remainder of 14.2.4 is inapplicable.
- d) When no gradient stop has the offset value of `0.0f`, then, defining n to be the offset value that is nearest to `0.0f` among the offset values in the set of all gradient stops, if o is in the offset range $[0, n)$, o 's color value shall be `rgba_color::transparent_black` and the remainder of these rules are inapplicable. [*Note*: Since the range described does not include n , it does not matter how many gradient stops have n as their offset value for purposes of this rule. — *end note*]
- e) When no gradient stop has the offset value of `1.0f`, then, defining n to be the offset value that is nearest to `1.0f` among the offset values in the set of all gradient stops, if o is in the offset range $(n, 1]$, o 's color value shall be `rgba_color::transparent_black` and the remainder of these rules are inapplicable. [*Note*: Since the range described does not include n , it does not matter how many gradient stops have n as their offset value for purposes of this rule. — *end note*]
- f) Each gradient stop has, at most, two adjacent gradient stops: one to its left and one to its right.
- g) Adjacency of gradient stops is initially determined by offset values. If two or more gradient stops have the same offset value then index values are used to determine adjacency as described below.
- h) For each gradient stop a , the *set of gradient stops to its left* are those gradient stops which have an offset value which is closer to `0.0f` than a 's offset value. [*Note*: This includes any gradient stops with an offset value of `0.0f` provided that a 's offset value is not `0.0f`. — *end note*]
- i) For each gradient stop b , the *set of gradient stops to its right* are those gradient stops which have an offset value which is closer to `1.0f` than b 's offset value. [*Note*: This includes any gradient stops with an offset value of `1.0f` provided that b 's offset value is not `1.0f`. — *end note*]
- j) A gradient stop which has an offset value of `0.0f` does not have an adjacent gradient stop to its left.
- k) A gradient stop which has an offset value of `1.0f` does not have an adjacent gradient stop to its right.
- l) If a gradient stop a 's set of gradient stops to its left consists of exactly one gradient stop, that gradient stop is the gradient stop that is adjacent to a on its left.
- m) If a gradient stop b 's set of gradient stops to its right consists of exactly one gradient stop, that gradient stop is the gradient stop that is adjacent to b on its right.
- n) If two or more gradient stops have the same offset value then the gradient stop with the lowest index value is the only gradient stop from that set of gradient stops which can have a gradient stop that is adjacent to it on its left and the gradient stop with the highest index value is the only gradient stop from that set of gradient stops which can have a gradient stop that is adjacent to it on its right. This rule takes precedence over all of the remaining rules.
- o) If a gradient stop can have an adjacent gradient stop to its left, then the gradient stop which is adjacent to it to its left is the gradient stop from the set of gradient stops to its left which has an offset value which is closest to its offset value. If two or more gradient stops meet that criteria, then the gradient stop which is adjacent to it to its left is the gradient stop which has the highest index value from the set of gradient stops to its left which are tied for being closest to its offset value.
- p) If a gradient stop can have an adjacent gradient stop to its right, then the gradient stop which is adjacent to it to its right is the gradient stop from the set of gradient stops to its right which has an offset value which is closest to its offset value. If two or more gradient stops meet that criteria, then the gradient stop which is adjacent to it to its right is the gradient stop which has the lowest index value from the set of gradient stops to its right which are tied for being closest to its offset value.
- q) Where the value of o is in the range $[0, 1]$, its color value shall be determined by interpolating between the gradient stop, r , which is the gradient stop whose offset value is closest to o without being less than

o and which can have an adjacent gradient stop to its left, and the gradient stop that is adjacent to r on r 's left. The acceptable forms of interpolating between color values is set forth later in this section.

r) Where the value of o is outside the range $[0, 1]$, its color value depends on the value of wrap mode:

- (1.1) — If wrap mode is `wrap_mode::none`, the color value of o shall be `rgba_color::transparent_black`.
- (1.2) — If wrap mode is `wrap_mode::pad`, if o is negative then the color value of o shall be the same as-if the value of o was `0.0f`, otherwise the color value of o shall be the same as-if the value of o was `1.0f`.
- (1.3) — If wrap mode is `wrap_mode::repeat`, then `1.0f` shall be added to or subtracted from o until o is in the range $[0, 1]$, at which point its color value is the color value for the modified value of o as determined by these rules. [*Example*: Given $o == 2.1$, after application of this rule $o == 0.1$ and the color value of o shall be the same value as-if the initial value of o was 0.1.
Given $o == -0.3$, after application of this rule $o == 0.7$ and the color value of o shall be the same as-if the initial value of o was 0.7. — *end example*]
- (1.4) — If wrap mode is `wrap_mode::reflect`, o shall be set to the absolute value of o , then `2.0f` shall be subtracted from o until o is in the range $[0, 2]$, then if o is in the range $(1, 2]$ then o shall be set to `1.0f - (o - 1.0f)`, at which point its color value is the color value for the modified value of o as determined by these rules. [*Example*: Given $o == 2.8$, after application of this rule $o == 0.8$ and the color value of o shall be the same value as-if the initial value of o was 0.8.
Given $o == 3.6$, after application of this rule $o == 0.4$ and the color value of o shall be the same value as-if the initial value of o was 0.4.
Given $o == -0.3$, after application of this rule $o == 0.3$ and the color value of o shall be the same as-if the initial value of o was 0.3.
Given $o == -5.8$, after application of this rule $o == 0.2$ and the color value of o shall be the same as-if the initial value of o was 0.2. — *end example*]

² Interpolation between the color values of two adjacent gradient stops is performed linearly on each color channel.

14.3 Enum class `wrap_mode` [io2d.wrapmode]

14.3.1 `wrap_mode` summary [io2d.wrapmode.summary]

- ¹ The `wrap_mode` enum class describes how a point's visual data is determined if it is outside the bounds of the *source brush* (16.3.2.2) when sampling.
- ² Depending on the source brush's `filter` value, the visual data of several points may be required to determine the appropriate visual data value for the point that is being sampled. In this case, each point is sampled according to the source brush's `wrap_mode` value with two exceptions:
1. If the point to be sampled is within the bounds of the source brush and the source brush's `wrap_mode` value is `wrap_mode::none`, then if the source brush's `filter` value requires that one or more points which are outside of the bounds of the source brush be sampled, each of those points is sampled as-if the source brush's `wrap_mode` value is `wrap_mode::pad` rather than `wrap_mode::none`.
 2. If the point to be sampled is within the bounds of the source brush and the source brush's `wrap_mode` value is `wrap_mode::none`, then if the source brush's `filter` value requires that one or more points which are inside of the bounds of the source brush be sampled, each of those points is sampled such that the visual data that is returned is the equivalent of `rgba_color::transparent_black`.
- ³ If a point to be sampled does not have a defined visual data element and the search for the nearest point with defined visual data produces two or more points with defined visual data that are equidistant from the point to be sampled, the returned visual data shall be an unspecified value which is the visual data of one of those equidistant points. Where possible, implementations should choose the among the equidistant points that have an x axisvalue and a y axisvalue that is nearest to `0.0f`.
- ⁴ See Table 27 for the meaning of each `wrap_mode` enumerator.

14.3.2 `wrap_mode` synopsis [io2d.wrapmode.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class wrap_mode {
        none,
```

```

repeat,
reflect,
pad
};
}

```

14.3.3 wrap_mode enumerators

[io2d.wrapmode.enumerators]

Table 27 — wrap_mode enumerator meanings

Enumerator	Meaning
none	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned is the equivalent of <code>rgba_color::transparent_black</code> .
repeat	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned is the visual data that would have been returned if the source brush was infinitely large and repeated itself in a left-to-right-left-to-right and top-to-bottom-top-to-bottom fashion.
reflect	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned is the visual data that would have been returned if the source brush was infinitely large and repeated itself in a left-to-right-to-left-to-right and top-to-bottom-to-top-to-bottom fashion.
pad	If the point to be sampled is outside of the bounds of the source brush, the visual data that is returned is the visual data that would have been returned for the nearest defined point that is in inside the bounds of the source brush.

14.4 Enum class filter

[io2d.filter]

14.4.1 filter summary

[io2d.filter.summary]

- The `filter` enum class specifies the type of filter to use when sampling from raster graphics data.
- 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.
- [*Note*: The only type of brush that has raster graphics data as its visual data is a brush with a brush type of `brush_type::surface`. — *end note*]
- See Table 28 for the meaning of each `filter` enumerator.

14.4.2 filter synopsis

[io2d.filter.synopsis]

```

namespace std::experimental::io2d::v1 {
enum class filter {
    fast,
    good,
    best,
    nearest,
    bilinear
};
}

```

14.4.3 filter enumerators

[io2d.filter.enumerators]

Table 28 — filter enumerator meanings

Enumerator	Meaning
fast	The filter that corresponds to this value is implementation-defined. The implementation shall ensure that the time complexity of the chosen filter is not greater than the time complexity of the filter that corresponds to <code>filter::good</code> . [<i>Note</i> : By choosing this value, the user is hinting that performance is more important than quality. — <i>end note</i>]
good	The filter that corresponds to this value is implementation-defined. The implementation shall ensure that the time complexity of the chosen formula is not greater than the time complexity of the formula for <code>filter::best</code> . [<i>Note</i> : By choosing this value, the user is hinting that quality and performance are equally important. — <i>end note</i>]
best	The filter that corresponds to this value is implementation-defined. [<i>Note</i> : By choosing this value, the user is hinting that quality is more important than performance. — <i>end note</i>]
nearest	Nearest-neighbor interpolation filtering
bilinear	Bilinear interpolation filtering

14.5 Enum class `brush_type`

[io2d.brushtype]

14.5.1 `brush_type` summary

[io2d.brushtype.summary]

- ¹ The `brush_type` enum class denotes the type of a brush object.
- ² See Table 29 for the meaning of each `brush_type` enumerator.

14.5.2 `brush_type` synopsis

[io2d.brushtype.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class brush_type {
        solid_color,
        surface,
        linear,
        radial
    };
}
```

14.5.3 `brush_type` enumerators

[io2d.brushtype.enumerators]

Table 29 — `brush_type` enumerator meanings

Enumerator	Meaning
<code>solid_color</code>	The brush object is a solid color brush.
<code>surface</code>	The brush object is a surface brush.
<code>linear</code>	The brush object is a linear gradient brush.
<code>radial</code>	The brush object is a radial gradient brush.

14.6 Class `gradient_stop`

[io2d.gradientstop]

14.6.1 Overview

[io2d.gradientstop.intro]

- ¹ The class `gradient_stop` describes a gradient stop that is used by gradient brushes.
- ² It has an *offset* of type `float` and an *offset_color* of type `rgba_color`.

14.6.2 gradient_stop synopsis**[io2d.gradientstop.synopsis]**

```

namespace std::experimental::io2d::v1 {
    class gradient_stop {
    public:
        // 14.6.3, construct:
        constexpr gradient_stop() noexcept;
        constexpr gradient_stop(float o, rgba_color c) noexcept;

        // 14.6.4, modifiers:
        constexpr void offset(float o) noexcept;
        constexpr void color(rgba_color c) noexcept;

        // 14.6.5, observers:
        constexpr float offset() const noexcept;
        constexpr rgba_color color() const noexcept;
    };
    // 14.6.6, operators:
    constexpr bool operator==(const gradient_stop& lhs, const gradient_stop& rhs)
        noexcept;
    constexpr bool operator!=(const gradient_stop& lhs, const gradient_stop& rhs)
        noexcept;
}

```

14.6.3 gradient_stop constructors**[io2d.gradientstop.cons]**

```
constexpr gradient_stop() noexcept;
```

1 *Effects:* Equivalent to: `gradient_stop(0.0f, rgba_color::transparent_black)`.

```
constexpr gradient_stop(float o, rgba_color c) noexcept;
```

2 *Requires:* `o >= 0.0f` and `o <= 1.0f`.

3 *Effects:* Constructs a `gradient_stop` object.

4 The offset is `o` rounded to the nearest multiple of `0.00001f`. The offset color is `c`.

14.6.4 gradient_stop modifiers**[io2d.gradientstop.modifiers]**

```
constexpr void offset(float o) noexcept;
```

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

2 *Effects:* The offset is `o` rounded to the nearest multiple of `0.00001f`.

```
constexpr void color(rgba_color c) noexcept;
```

3 *Effects:* The offset color is `c`.

14.6.5 gradient_stop observers**[io2d.gradientstop.observers]**

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

1 *Returns:* The offset.

```
constexpr rgba_color color() const noexcept;
```

2 *Returns:* The offset color.

14.6.6 gradient_stop operators**[io2d.gradientstop.ops]**

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

1 *Returns:* `lhs.offset() == rhs.offset() && lhs.color() == rhs.color()`;

14.7 Class template basic_brush**[io2d.brush]****14.7.1 Summary****[io2d.brush.intro]**

1 The class template `basic_brush` describes an opaque wrapper for visual data. It takes one type parameter, which is a `GraphicsSurfaces`.

- 2 A `basic_brush` object is usable with any `basic_image_surface`, `basic_output_surface`, and `basic_unmanaged_output_surface` object provided that they have the same `GraphicsSurfaces` as the `basic_brush` object.
- 3 A `basic_brush` object's visual data is immutable. It is observable only by the effect that it produces when the brush is used as a *source brush* or as a *mask brush* (16.3.2.2).
- 4 A `basic_brush` object has a brush type of `brush_type`, which indicates which type of brush it is (Table 29).
- 5 As a result of technological limitations, a `basic_brush` object's visual data may have less precision than the data from which it was created.
- 6 The data are stored in an object of type `typename GraphicsMath::brushes::brush_data_type`.

14.7.2 Synopsis

[io2d.brush.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_brush {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::brushes::brush_data_type;

        // 14.7.4, constructors:
        explicit basic_brush(rgba_color c);
        template <class InputIterator>
        basic_brush(const basic_point_2d<graphics_math_type>& begin,
                   const basic_point_2d<graphics_math_type>& end,
                   InputIterator first, InputIterator last);
        basic_brush(const basic_point_2d<graphics_math_type>& begin,
                   const basic_point_2d<graphics_math_type>& end,
                   initializer_list<gradient_stop> il);
        template <class InputIterator>
        basic_brush(const basic_circle<graphics_math_type>& start,
                   const basic_circle<graphics_math_type>& end,
                   InputIterator first, InputIterator last);
        basic_brush(const basic_circle<graphics_math_type>& start,
                   const basic_circle<graphics_math_type>& end,
                   initializer_list<gradient_stop> il);
        basic_brush(basic_image_surface<GraphicsSurfaces>&& img);

        // 14.7.5, accessors:
        const data_type& data() const noexcept;
        brush_type type() const noexcept;
    };
}
```

14.7.3 Sampling from a `basic_brush` object

[io2d.brush.sampling]

- 1 A `basic_brush` object is sampled from either as a source brush (16.3.2.2) or a mask brush (16.3.2.2).
- 2 If it is being sampled from as a source brush, its *wrap mode*, *filter*, and *brush matrix* are defined by a `basic_brush_props` object (16.3.2.4 and 16.3.2.6).
- 3 If it is being sampled from as a mask brush, its *wrap mode*, *filter*, and *mask matrix* are defined by a `basic_mask_props` object (16.3.2.5 and 16.3.2.6).
- 4 When sampling from a `basic_brush` object `b`, the `brush_type` returned by calling `b.type()` determines how the results of sampling are determined:
 1. If the result of `b.type()` is `brush_type::solid_color` then `b` is a *solid color brush*.
 2. If the result of `b.type()` is `brush_type::surface` then `b` is a *surface brush*.
 3. If the result of `b.type()` is `brush_type::linear` then `b` is a *linear gradient brush*.
 4. If the result of `b.type()` is `brush_type::radial` then `b` is a *radial gradient brush*.

14.7.3.1 Sampling from a solid color brush [io2d.brush.sampling.color]

- ¹ When **b** is a solid color brush, then when sampling from **b**, the visual data returned is always the visual data used to construct **b**, regardless of the point which is to be sampled and regardless of the return values of wrap mode, filter, and brush matrix or mask matrix.

14.7.3.2 Sampling from a linear gradient brush [io2d.brush.sampling.linear]

- ¹ When **b** is a linear gradient brush, when sampling point **pt**, where **pt** is the return value of calling the `transform_pt` member function of brush matrix or mask matrix using the requested point, from **b**, the visual data returned are as specified by [14.2.2](#) and [14.2.4](#).

14.7.3.3 Sampling from a radial gradient brush [io2d.brush.sampling.radial]

- ¹ When **b** is a radial gradient brush, when sampling point **pt**, where **pt** is the return value of calling the `transform_pt` member function of brush matrix or mask matrix using the requested point, from **b**, the visual data are as specified by [14.2.3](#) and [14.2.4](#).

14.7.3.4 Sampling from a surface brush [io2d.brush.sampling.surface]

- ¹ When **b** is a surface brush, when sampling point **pt** from **b**, where **pt** is the return value of calling the `transform_pt` member function of the brush matrix or mask matrix using the requested point, the visual data returned are from the point **pt** in the raster graphics data of the brush, as modified by the values of wrap mode ([14.3](#)) and filter ([14.4](#)).

14.7.4 Constructors [io2d.brush.ctor]

```
explicit basic_brush(rgba_color c);
```

- ¹ *Effects:* Constructs an object of type `basic_brush`.

² *Postconditions:* `data() == GraphicsSurfaces::brushes::create_brush(c)`.

³ *Remarks:* The visual data format of the visual data are as-if it is that specified by `format::argb32`.

⁴ Sampling from the brush produces the results specified in [14.7.3.1](#).

```
template <class InputIterator>
basic_brush(const basic_point_2d<graphics_math_type>& begin,
            const basic_point_2d<graphics_math_type>& end,
            InputIterator first, InputIterator last);
```

- ⁵ *Effects:* Constructs an object of type `basic_brush`.

⁶ *Postconditions:* `data() == GraphicsSurfaces::brushes::create_brush(begin, end, first, last)`.

⁷ *Remarks:* Sampling from this brush produces the results specified in [14.7.3.2](#).

```
basic_brush(const basic_point_2d<graphics_math_type>& begin,
            const basic_point_2d<graphics_math_type>& end,
            initializer_list<gradient_stop> il);
```

- ⁸ *Effects:* Constructs an object of type `basic_brush`.

⁹ *Postconditions:* `data() == GraphicsSurfaces::brushes::create_brush(begin, end, il)`.

¹⁰ *Remarks:* Sampling from this brush produces the results specified in [14.7.3.2](#).

```
template <class InputIterator>
basic_brush(const basic_circle<graphics_math_type>& start,
            const basic_circle<graphics_math_type>& end,
            InputIterator first, InputIterator last);
```

- ¹¹ *Effects:* Constructs an object of type `basic_brush`.

¹² *Postconditions:* `data() == GraphicsSurfaces::brushes::create_brush(start, end, first, last)`.

¹³ *Remarks:* Sampling from this brush produces the results specified in [14.7.3.3](#).

```
basic_brush(const basic_circle<graphics_math_type>& start,
            const basic_circle<graphics_math_type>& end,
            initializer_list<gradient_stop> il);
```

- ¹⁴ *Effects:* Constructs an object of type `basic_brush`.

15 *Postconditions:* `data() == GraphicsSurfaces::brushes::create_brush(start, end, il)`.

16 *Remarks:* Sampling from this brush produces the results specified in [14.7.3.3](#).

```
basic_brush(basic_image_surface<GraphicsSurfaces>&& img);
```

17 *Effects:* Constructs an object of type `basic_brush`.

18 *Postconditions:* `data() == GraphicsSurfaces::brushes::create_brush(move(img))`.

19 Sampling from this brush produces the results specified in [14.7.3.4](#).

14.7.5 Accessors

[io2d.brush.acc]

```
const data_type& data() const noexcept;
```

1 *Returns:* A reference to the `basic_brush` object's data object (See [14.7.1](#)).

```
brush_type type() const noexcept;
```

2 *Returns:* `GraphicsSurfaces::brushes::get_brush_type(data())`.

15 Surface state props [io2d.surfacestate]

15.1 Overview [io2d.surfacestate.general]

- ¹ In order to produce effects beyond simply drawing raster graphics data or a path to a surface, graphics state data is supplied when performing rendering and compositing operations (16.3.2) on surfaces.
- ² *Surface state* types group together related graphics state data. Objects of those types are then supplied as arguments to the functions that carry out rendering and compositing operations on surfaces. [*Note*: This allows surfaces to be stateless, which typically provides significant performance gains on modern graphics acceleration hardware. — *end note*]
- ³ The `enum class` types and surface state class templates that define and provide the graphics state data are described below.

15.2 Enum class `antialias` [io2d.antialias]

15.2.1 `antialias` summary [io2d.antialias.summary]

- ¹ The `antialias` enum class specifies the type of anti-aliasing that the rendering system uses for rendering paths. See Table 30 for the meaning of each `antialias` enumerator.

15.2.2 `antialias` synopsis [io2d.antialias.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class antialias {
        none,
        fast,
        good,
        best
    };
}
```

15.2.3 `antialias` enumerators [io2d.antialias.enumerators]

Table 30 — `antialias` enumerator meanings

Enumerator	Meaning
<code>none</code>	No anti-aliasing is performed when performing a rendering operation.
<code>fast</code>	Some form of anti-aliasing should be used when performing a rendering operation but performance is more important than the quality of the results. The technique used is implementation-defined.
<code>good</code>	Some form of anti-aliasing should be used when performing a rendering operation and the sacrificing some performance to obtain better anti-aliasing results than would likely be obtained from <code>antialias::fast</code> is acceptable. The technique used is implementation-defined.
<code>best</code>	Some form of anti-aliasing should be used when performing a rendering operation and better anti-aliasing results than would likely be obtained from <code>antialias::fast</code> and <code>antialias::good</code> are desired even if performance degrades significantly. The technique used is implementation-defined. [<i>Note</i> : This might commonly be chosen when a user is going to render something once and cache the results for repeated use or when a user is rendering something that does not necessarily need performance suitable for real-time computer graphics applications. — <i>end note</i>]

15.3 Enum class `fill_rule` [io2d.fillrule]

15.3.1 `fill_rule` summary [io2d.fillrule.summary]

- ¹ The `fill_rule` enum class determines how the filling operation (16.3.5) is performed on a path.
- ² For each point, draw a ray from that point to infinity which does not pass through the start point or end point of any non-degenerate segment in the path, is not tangent to any non-degenerate segment in the path, and is not coincident with any non-degenerate segment in the path.
- ³ See Table 31 for the meaning of each `fill_rule` enumerator.

15.3.2 `fill_rule` synopsis [io2d.fillrule.synopsis]

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

15.3.3 `fill_rule` enumerators [io2d.fillrule.enumerators]

Table 31 — `fill_rule` enumerator meanings

Enumerator	Meaning
<code>winding</code>	If the <i>fill rule</i> (15.10.1) is <code>fill_rule::winding</code> , then using the ray described above and beginning with a count of zero, add one to the count each time a non-degenerate segment crosses the ray going left-to-right from its begin point to its end point, and subtract one each time a non-degenerate segment crosses the ray going from right-to-left from its begin point to its end point. If the resulting count is zero after all non-degenerate segments that cross the ray have been evaluated, the point shall not be filled; otherwise the point shall be filled.
<code>even_odd</code>	If the fill rule is <code>fill_rule::even_odd</code> , then using the ray described above and beginning with a count of zero, add one to the count each time a non-degenerate segment crosses the ray. If the resulting count is an odd number after all non-degenerate segments that cross the ray have been evaluated, the point shall be filled; otherwise the point shall not be filled. [<i>Note</i> : Mathematically, zero is an even number, not an odd number. — <i>end note</i>]

15.4 Enum class `line_cap` [io2d.linecap]

15.4.1 `line_cap` summary [io2d.linecap.summary]

- ¹ The `line_cap` enum class specifies how the ends of lines should be rendered when a `interpreted_path` object is stroked. See Table 32 for the meaning of each `line_cap` enumerator.

15.4.2 `line_cap` synopsis [io2d.linecap.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class line_cap {
        none,
        round,
        square
    };
}
```

15.4.3 `line_cap` enumerators [io2d.linecap.enumerators]

Table 32 — `line_cap` enumerator meanings

Enumerator	Meaning
<code>none</code>	The line has no cap. It terminates exactly at the end point.
<code>round</code>	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.
<code>square</code>	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.

15.5 Enum class `line_join` [io2d.linejoin]

15.5.1 `line_join` summary [io2d.linejoin.summary]

- ¹ The `line_join` enum class specifies how the junction of two line segments should be rendered when a `interpreted_path` is stroked. See Table 33 for the meaning of each enumerator.

15.5.2 `line_join` synopsis [io2d.linejoin.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class line_join {
        miter,
        round,
        bevel
    };
}
```

15.5.3 `line_join` enumerators [io2d.linejoin.enumerators]

Table 33 — `line_join` enumerator meanings

Enumerator	Meaning
<code>miter</code>	Joins will be mitered or beveled, depending on the miter limit (see: 15.12.1).
<code>round</code>	Joins will be rounded, with the center of the circle being the join point.
<code>bevel</code>	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.

15.6 Enum class `compositing_op` [io2d.compositingop]

15.6.1 `compositing_op` Summary [io2d.compositingop.summary]

- ¹ The `compositing_op` enum class specifies composition algorithms. See Table 34, Table 35 and Table 36 for the meaning of each `compositing_op` enumerator.

15.6.2 `compositing_op` Synopsis [io2d.compositingop.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class compositing_op {
        // basic
        over,
        clear,
        source,
        in,
        out,
        atop,
        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
};
}

```

15.6.3 compositing_op Enumerators

[io2d.compositingop.enumerators]

- 1 The tables below specifies the mathematical formula for each enumerator's composition algorithm. The formulas differentiate between three color channels (red, green, and blue) and an alpha channel (transparency). For all channels, valid channel values are in the range [0.0, 1.0].
- 2 Where a visual data format for a visual data element has no alpha channel, the visual data format shall be treated as though it had an alpha channel with a value of 1.0 for purposes of evaluating the formulas.
- 3 Where a visual data format for a visual data element has no color channels, the visual data format shall be treated as though it had a value of 0.0 for all color channels for purposes of evaluating the formulas.
- 4 The following symbols and specifiers are used:
 - The *R* symbol means the result color value
 - The *S* symbol means the source color value
 - The *D* symbol means the destination color value
 - The *c* specifier means the color channels of the value it follows
 - The *a* specifier means the alpha channel of the value it follows
- 5 The color symbols *R*, *S*, and *D* may appear with or without any specifiers.
- 6 If a color symbol appears alone, it designates the entire color as a tuple in the unsigned normalized form (red, green, blue, alpha).
- 7 The specifiers *c* and *a* may appear alone or together after any of the three color symbols.
- 8 The presence of the *c* specifier alone means the three color channels of the color as a tuple in the unsigned normalized form (red, green, blue).
- 9 The presence of the *a* specifier alone means the alpha channel of the color in unsigned normalized form.
- 10 The presence of the specifiers together in the form *ca* means the value of the color as a tuple in the unsigned normalized form (red, green, blue, alpha), where the value of each color channel is the product of each color channel and the alpha channel and the value of the alpha channel is the original value of the alpha channel. [*Example:* When it appears in a formula, *Sca* means $((Sc \times Sa), Sa)$, such that, given a source color $Sc = (1.0, 0.5, 0.0)$ and an source alpha $Sa = (0.5)$, the value of *Sca* when specified in one of the formulas would be $Sca = (1.0 \times 0.5, 0.5 \times 0.5, 0.0 \times 0.5, 0.5) = (0.5, 0.25, 0.0, 0.5)$. The same is true for *Dca* and *Rca*. — *end example*]
- 11 No space is left between a value and its channel specifiers. Channel specifiers will be preceded by exactly one value symbol.
- 12 When performing an operation that involves evaluating the color channels, each color channel should be evaluated individually to produce its own value.

- 13 The basic enumerators specify a value for *bound*. This value may be 'Yes', 'No', or 'N/A'.
- 14 If the bound value is 'Yes', then the source is treated as though it is also a mask. As such, only areas of the surface where the source would affect the surface are altered. The remaining areas of the surface have the same color value as before the compositing operation.
- 15 If the bound value is 'No', then every area of the surface that is not affected by the source will become transparent black. In effect, it is as though the source was treated as being the same size as the destination surface with every part of the source that does not already have a color value assigned to it being treated as though it were transparent black. Application of the formula with this precondition results in those areas evaluating to transparent black such that evaluation can be bypassed due to the predetermined outcome.
- 16 If the bound value is 'N/A', the operation would have the same effect regardless of whether it was treated as 'Yes' or 'No' such that those bound values are not applicable to the operation. A 'N/A' formula when applied to an area where the source does not provide a value will evaluate to the original value of the destination even if the source is treated as having a value there of transparent black. As such the result is the same as-if the source were treated as being a mask, i.e. 'Yes' and 'No' treatment each produce the same result in areas where the source does not have a value.
- 17 If a clip is set and the bound value is 'Yes' or 'N/A', then only those areas of the surface that the are within the clip will be affected by the compositing operation.
- 18 If a clip is set and the bound value is 'No', then only those areas of the surface that the are within the clip will be affected by the compositing operation. Even if no part of the source is within the clip, the operation will still set every area within the clip to transparent black. Areas outside the clip are not modified.

Table 34 — `compositing_op` basic enumerator meanings

Enumerator	Bound	Color	Alpha
<code>clear</code>	Yes	$Rc = 0$	$Ra = 0$
<code>source</code>	Yes	$Rc = Sc$	$Ra = Sa$
<code>over</code>	N/A	$Rc = \frac{(Sca + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da \times (1 - Sa)$
<code>in</code>	No	$Rc = Sc$	$Ra = Sa \times Da$
<code>out</code>	No	$Rc = Sc$	$Ra = Sa \times (1 - Da)$
<code>atop</code>	N/A	$Rc = Sca + Dc \times (1 - Sa)$	$Ra = Da$
<code>dest_over</code>	N/A	$Rc = \frac{(Sca \times (1 - Da) + Dca)}{Ra}$	$Ra = (1 - Da) \times Sa + Da$
<code>dest_in</code>	No	$Rc = Dc$	$Ra = Sa \times Da$
<code>dest_out</code>	N/A	$Rc = Dc$	$Ra = (1 - Sa) \times Da$
<code>dest_atop</code>	No	$Rc = Sc \times (1 - Da) + Dca$	$Ra = Sa$
<code>xor_op</code>	N/A	$Rc = \frac{(Sca \times (1 - Da) + Dca \times (1 - Sa))}{Ra}$	$Ra = Sa + Da - 2 \times Sa \times Da$
<code>add</code>	N/A	$Rc = \frac{(Sca + Dca)}{Ra}$	$Ra = \min(1, Sa + Da)$
<code>saturate</code>	N/A	$Rc = \frac{(\min(Sa, 1 - Da) \times Sc + Dca)}{Ra}$	$Ra = \min(1, Sa + Da)$

- 19 The blend enumerators and hsl enumerators share a common formula for the result color's color channel, with only one part of it changing depending on the enumerator. The result color's color channel value formula is as follows: $Rc = \frac{1}{Ra} \times ((1 - Da) \times Sca + (1 - Sa) \times Dca + Sa \times Da \times f(Sc, Dc))$. The function $f(Sc, Dc)$ is the component of the formula that is enumerator dependent.
- 20 For the blend enumerators, the color channels shall be treated as separable, meaning that the color formula shall be evaluated separately for each color channel: red, green, and blue.

- 21 The color formula divides 1 by the result color's alpha channel value. As a result, if the result color's alpha channel is zero then a division by zero would normally occur. Implementations shall not throw an exception nor otherwise produce any observable error condition if the result color's alpha channel is zero. Instead, implementations shall bypass the division by zero and produce the result color (0, 0, 0, 0), i.e. *transparent black*, if the result color alpha channel formula evaluates to zero. [*Note*: The simplest way to comply with this requirement is to bypass evaluation of the color channel formula in the event that the result alpha is zero. However, in order to allow implementations the greatest latitude possible, only the result is specified. — *end note*]
- 22 For the enumerators in Table 35 and Table 36 the result color's alpha channel value formula is as follows: $Ra = Sa + Da \times (1 - Sa)$. [*Note*: Since it is the same formula for all enumerators in those tables, the formula is not included in those tables. — *end note*]
- 23 All of the blend enumerators and hsl enumerators have a bound value of 'N/A'.

Table 35 — `compositing_op` blend enumerator meanings

Enumerator	Color
<code>multiply</code>	$f(Sc, Dc) = Sc \times Dc$
<code>screen</code>	$f(Sc, Dc) = Sc + Dc - Sc \times Dc$
<code>overlay</code>	$\text{if}(Dc \leq 0.5f) \{$ $f(Sc, Dc) = 2 \times Sc \times Dc$ $\}$ $\text{else} \{$ $f(Sc, Dc) =$ $1 - 2 \times (1 - Sc) \times$ $(1 - Dc)$ $\}$ <p>[<i>Note</i>: The difference between this enumerator and <code>hard_light</code> is that this tests the destination color (Dc) whereas <code>hard_light</code> tests the source color (Sc). — <i>end note</i>]</p>
<code>darken</code>	$f(Sc, Dc) = \min(Sc, Dc)$
<code>lighten</code>	$f(Sc, Dc) = \max(Sc, Dc)$
<code>color_dodge</code>	$\text{if}(Dc < 1) \{$ $f(Sc, Dc) = \min(1, \frac{Dc}{(1 - Sc)})$ $\}$ $\text{else} \{$ $f(Sc, Dc) = 1\}$
<code>color_burn</code>	$\text{if}(Dc > 0) \{$ $f(Sc, Dc) = 1 - \min(1, \frac{1 - Dc}{Sc})$ $\}$ $\text{else} \{$ $f(Sc, Dc) = 0$ $\}$
<code>hard_light</code>	$\text{if}(Sc \leq 0.5f) \{$ $f(Sc, Dc) = 2 \times Sc \times Dc$ $\}$ $\text{else} \{$ $f(Sc, Dc) =$ $1 - 2 \times (1 - Sc) \times$ $(1 - Dc)$ $\}$ <p>[<i>Note</i>: The difference between this enumerator and <code>overlay</code> is that this tests the source color (Sc) whereas <code>overlay</code> tests the destination color (Dc). — <i>end note</i>]</p>

Table 35 — `compositing_op` blend enumerator meanings (continued)

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

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

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

26 $\text{min}(x, y, z) = \text{min}(x, \text{min}(y, z))$

27 $\text{max}(x, y, z) = \text{max}(x, \text{max}(y, z))$

28 $\text{sat}(C) = \text{max}(Cr, Cg, Cb) - \text{min}(Cr, Cg, Cb)$

29 $\text{lum}(C) = Cr \times 0.3 + Cg \times 0.59 + Cb \times 0.11$

30 $\text{clip_color}(C) = \{$

$$L = \text{lum}(C)$$

$$N = \text{min}(Cr, Cg, Cb)$$

$$X = \text{max}(Cr, Cg, Cb)$$

$$\text{if } (N < 0.0) \{$$

$$Cr = L + \frac{((Cr - L) \times L)}{(L - N)}$$

$$Cg = L + \frac{((Cg - L) \times L)}{(L - N)}$$

$$Cb = L + \frac{((Cb - L) \times L)}{(L - N)}$$

$$\}$$

$$\text{if } (X > 1.0) \{$$

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

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

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

$$\}$$

```

    return C
}
31 set_lum(C, L) = {
    D = L - lum(C)
    Cr = Cr + D
    Cg = Cg + D
    Cb = Cb + D
    return clip_color(C)
}
32 set_sat(C, S) = {
    R = C
    auto& max = (Rr > Rg) ? ((Rr > Rb) ? Rr : Rb) : ((Rg > Rb) ? Rg : Rb)
    auto& mid = (Rr > Rg) ? ((Rr > Rb) ? ((Rg > Rb) ? Rg : Rb) : Rr) : ((Rg > Rb) ? ((Rr > Rb) ? Rr :
Rb) : Rg)
    auto& min = (Rr > Rg) ? ((Rg > Rb) ? Rb : Rg) : ((Rr > Rb) ? Rb : Rr)
    if (max > min) {
        mid = ((mid - min) × S) / (max - min)
        max = S
    }
    else {
        mid = 0.0
        max = 0.0
    }
    min = 0.0
    return R
}

```

[*Note:* In the formula, *max*, *mid*, and *min* are reference variables which are bound to the highest value, second highest value, and lowest value color channels of the (red, blue, green) tuple *R* such that the subsequent operations modify the values of *R* directly. — *end note*]

Table 36 — `compositing_op` hsl enumerator meanings

Enumerator	Color & Alpha
<code>hsl_hue</code>	$f(Sc, Dc) = set_lum(set_sat(Sc, sat(Dc)), lum(Dc))$
<code>hsl_saturation</code>	$(Sc, Dc) = set_lum(set_sat(Dc, sat(Sc)), lum(Dc))$
<code>hsl_color</code>	$f(Sc, Dc) = set_lum(Sc, lum(Dc))$
<code>hsl_luminosity</code>	$f(Sc, Dc) = set_lum(Dc, lum(Sc))$

15.7 Enum class format [io2d.format]

15.7.1 Summary [io2d.format.summary]

- ¹ The `format` enum class indicates a visual data format. See Table 37 for the meaning of each format enumerator.

15.7.2 Synopsis [io2d.format.synopsis]

```

namespace std::experimental::io2d::v1 {
    enum class format {
        invalid,
        argb32,
        xrgb32,
        xrgb16,
        a8
    };
}

```

15.7.3 Enumerators [io2d.format.enumerators]

Table 37 — format enumerator meanings

Enumerator	Meaning
<code>invalid</code>	A previously requested <code>format</code> is unsupported by the implementation.
<code>argb32</code>	A 32-bit RGB color model pixel format. There is an 8 bit alpha channel, an 8-bit red color channel, an 8-bit green color channel, and an 8-bit blue color channel. The byte order, interpretation of values within each channel, and whether or not this is a premultiplied format are implementation-defined.
<code>xrgb32</code>	A 32-bit RGB color model pixel format. There is an 8 bit channel that is not used, an 8-bit red color channel, an 8-bit green color channel, and an 8-bit blue color channel. The byte order and interpretation of values within each channel are implementation-defined.
<code>xrgb16</code>	A 16-bit RGB color model pixel format. There is a red color channel, a green color channel, and a blue color channel. The number of bits, byte order, and interpretation of values within each channel are implementation-defined.
<code>a8</code>	An 8-bit transparency data pixel format. All 8 bits are an alpha channel.

¹ Implementations may support additional visual data formats (See: 9.2.3).

15.8 Enum class scaling

[io2d.scaling]

15.8.1 scaling summary

[io2d.scaling.summary]

- ¹ The scaling enum class specifies the type of scaling an output surface will use when the size of its *display buffer* (16.3.9) differs from the size of its *back buffer* (16.3.9).
- ² See Table 38 for the meaning of each `scaling` enumerator.

15.8.2 scaling synopsis

[io2d.scaling.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class scaling {
        letterbox,
        uniform,
        fill_uniform,
        fill_exact,
        none
    };
}
```

15.8.3 scaling enumerators

[io2d.scaling.enumerators]

- ¹ [Note: In the following table, examples will be given to help explain the meaning of each enumerator. The examples will all use a `basic_output_surface` object called `ds`.

The back buffer (16.3.9) 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 (16.3.9) 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 $(x_1, y_1) = (10, 10)$, $(x_2, y_2) = (20, 20)$ 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 38 — scaling enumerator meanings

Enumerator	Meaning
letterbox	<p>Fill the display buffer with the letterbox brush (16.3.9) of the <code>basic_output_surface</code>. 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.</p> <p>[<i>Example:</i> The display buffer of <code>ds</code> will be filled with the brush object returned by <code>ds.letterbox_brush()</code>; . The back buffer of <code>ds</code> 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</p> $(x1, y1) = \left(\frac{1280}{2} - \frac{960}{2}, 0\right) = (160, 0),$ $(x2, y2) = \left(960 + \left(\frac{1280}{2} - \frac{960}{2}\right), 720\right) = (1120, 720).$ <p>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). — <i>end example</i>]</p>

Table 38 — `scaling` enumerator meanings (continued)

Enumerator	Meaning
uniform	<p>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.</p> <p>[<i>Example:</i> The back buffer of <code>ds</code> 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</p> $(x1, y1) = \left(\frac{1280}{2} - \frac{960}{2}, 0\right) = (160, 0),$ $(x2, y2) = \left(960 + \left(\frac{1280}{2} - \frac{960}{2}\right), 720\right) = (1120, 720).$ <p>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). — <i>end example</i>]</p> <p>[<i>Note:</i> The difference between <code>uniform</code> and <code>letterbox</code> is that <code>uniform</code> does not modify the contents of the display buffer that fall outside of the rectangle into which the scaled back buffer is drawn while <code>letterbox</code> fills those areas with the <code>basic_output_surface</code> object's <code>letterbox</code> brush (see: 16.3.9). — <i>end note</i>]</p>
fill_uniform	<p>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.</p> <p>[<i>Example:</i> The back buffer of <code>ds</code> will be drawn in the rectangle $(x1, y1) = (0, -120)$, $(x2, y2) = (1280, 840)$. This fulfills all of the conditions. At least one dimension of the scaled back buffer is the same length as the same dimension of the display buffer (both have a width of 1280 pixels). The second dimension of the scaled back buffer is not shorter than the second dimension of the display buffer (the back buffer's scaled height is 840 pixels, which is not shorter than the display buffer's height of 720 pixels). Lastly, the scaled back buffer is centered in the display buffer (on the x axis there are 0 pixels between each vertical side of the rectangle and the nearest vertical edge of the display buffer and on the y axis there are 120 pixels between each horizontal side of the rectangle and the nearest horizontal edge of the display buffer). — <i>end example</i>]</p>

Table 38 — `scaling` enumerator meanings (continued)

Enumerator	Meaning
<code>fill_exact</code>	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. [<i>Example:</i> The back buffer will be drawn in the rectangle $(x1, y1) = (0, 0)$, $(x2, y2) = (1280, 720)$. This fulfills all of the conditions. Each dimension of the scaled back buffer is the same length as the same dimension of the display buffer (both have a width of 1280 pixels and a height of 720 pixels) and the origin of the scaled back buffer is at the origin of the display buffer. — <i>end example</i>]
<code>none</code>	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. [<i>Example:</i> The back buffer of <code>ds</code> will be drawn in the rectangle $(x1, y1) = (0, 0)$, $(x2, y2) = (640, 480)$ such that no scaling occurs and the origin of the back buffer is at the origin of the display buffer. — <i>end example</i>]

15.9 Class template `basic_render_props`

[[io2d.renderprops](#)]

15.9.1 Overview

[[io2d.renderprops.intro](#)]

- ¹ The `basic_render_props` class template provides general state information that is applicable to all rendering and compositing operations ([16.3.2](#)).
- ² It has a *filter algorithm* of type `filter`, a *surface matrix* of type `basic_matrix_2d`, and a *compositing operator* of type `compositing_op`.
- ³ The data are stored in an object of type `typename GraphicsSurfaces::surface_state_props::render_props_data_type`. It is accessible using the data member functions.

15.9.2 Synopsis

[[io2d.renderprops.synopsis](#)]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_render_props {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surface_state_props::render_props_data_type;

        // 15.9.3, constructors:
        basic_render_props() noexcept;
        explicit basic_render_props(filter f,
            const basic_matrix_2d<graphics_math_type>& m = basic_matrix_2d<graphics_math_type>{},
            compositing_op co = compositing_op::over) noexcept;

        // 15.9.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 15.9.5, modifiers:
        void filtering(filter f) noexcept;
        void compositing(compositing_op co) noexcept;
        void surface_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;

        // 15.9.6, observers:
        filter filtering() const noexcept;
        compositing_op compositing() const noexcept;

```

```

    basic_matrix_2d<graphics_math_type> surface_matrix() const noexcept;
};
}

```

15.9.3 Constructors [io2d.renderprops.ctor]

```
basic_render_props() noexcept;
```

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

2 *Postconditions:* `data() == X::surface_state_props::create_render_props()`.

```
explicit basic_render_props(filter f,
    const basic_matrix_2d<graphics_math_type>& m = basic_matrix_2d<graphics_math_type>{},
    compositing_op co = compositing_op::over) noexcept;
```

3 *Requires:* `m.is_invertible() == true`.

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

5 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_render_props(f, m, co)`.

15.9.4 Accessors [io2d.renderprops.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_render_props` object's data object (See: [15.9.1](#)).

15.9.5 Modifiers [io2d.renderprops.mod]

```
void filtering(filter f) noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::surface_state_props::filtering(data(), f)`.

Remarks: The filtering algorithm is `f`.

```
void compositing(compositing_op co) noexcept;
```

2 *Effects:* Calls `GraphicsSurfaces::surface_state_props::compositing(data(), co)`.

Remarks: The compositing operator is `co`.

```
void surface_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
```

3 *Requires:* `m.is_invertible() == true`.

4 *Effects:* Calls `GraphicsSurfaces::surface_state_props::surface_matrix(data(), m)`.

5 *Remarks:* The surface matrix is `m`.

15.9.6 Observers [io2d.renderprops.obs]

```
filter filtering() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surface_state_props::filtering(data())`.

2 *Remarks:* The returned value is the filter algorithm.

```
compositing_op compositing() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surface_state_props::compositing(data())`.

4 *Remarks:* The returned value is the compositing operator.

```
basic_matrix_2d<graphics_math_type> surface_matrix() const noexcept;
```

5 *Returns:* `GraphicsSurfaces::surface_state_props::surface_matrix(data())`.

6 *Remarks:* The returned value is the surface matrix.

15.10 Class template `basic_brush_props` [io2d.brushprops]

15.10.1 `basic_brush_props` summary [io2d.brushprops.summary]

- ¹ The `basic_brush_props` class template provides general state information that is applicable to all rendering and composing operations (16.3.2).
- ² It has a *wrap mode* of type `wrap_mode`, a *filter* of type `filter`, and a *brush matrix* of type `basic_matrix_2d`.
- ³ The data are stored in an object of type `typename GraphicsSurfaces::surface_state_props::brush_props_data_type`. It is accessible using the data member functions.

15.10.2 `basic_brush_props` synopsis [io2d.brushprops.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_brush_props {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surface_state_props::brush_props_data_type;

        // 15.10.3, constructor:
        basic_brush_props(io2d::wrap_mode w = io2d::wrap_mode::none,
            io2d::filter fi = io2d::filter::good,
            const basic_matrix_2d<graphics_math_type>& m = basic_matrix_2d<graphics_math_type>{})
            noexcept;

        // 15.10.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 15.10.5, modifiers:
        void wrap_mode(io2d::wrap_mode w) noexcept;
        void filter(io2d::filter fi) noexcept;
        void brush_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;

        // 15.10.6, observers:
        io2d::wrap_mode wrap_mode() const noexcept;
        io2d::filter filter() const noexcept;
        basic_matrix_2d<graphics_math_type> brush_matrix() const noexcept;
    };
}
```

15.10.3 `basic_brush_props` constructor [io2d.brushprops.cons]

```
basic_brush_props(io2d::wrap_mode w = io2d::wrap_mode::none,
    io2d::filter fi = io2d::filter::good,
    const basic_matrix_2d<graphics_math_type>& m = basic_matrix_2d<graphics_math_type>{})
    noexcept;
```

- ¹ *Requires:* `m.is_invertible() == true`.
- ² *Effects:* Constructs an object of type `basic_brush_props`.
- ³ *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_brush_props(w, fi, m)`.

15.10.4 Accessors [io2d.brushprops.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

- ¹ *Returns:* A reference to the `basic_brush_props` object's data object (See: 15.10.1).

15.10.5 `basic_brush_props` modifiers [io2d.brushprops.modifiers]

```
void wrap_mode(io2d::wrap_mode w) noexcept;
```

- ¹ *Effects:* Calls `GraphicsSurfaces::surface_state_props::wrap_mode(data(), w)`.

2 *Remarks:* The wrap mode is w.

```
void filter(io2d::filter fi) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surface_state_props::filter(data(), fi)`.

4 *Remarks:* The filter is fi.

```
void brush_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
```

5 *Requires:* `m.is_invertible() == true`.

6 *Effects:* Calls `GraphicsSurfaces::surface_state_props::brush_matrix(data(), m)`.

7 *Remarks:* The brush matrix is m.

15.10.6 basic_brush_props observers [io2d.brushprops.observers]

```
io2d::wrap_mode wrap_mode() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surface_state_props::wrap_mode(data())`.

```
io2d::filter filter() const noexcept;
```

2 *Returns:* `GraphicsSurfaces::surface_state_props::filter(data())`.

```
basic_matrix_2d<graphics_math_type> brush_matrix() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surface_state_props::brush_matrix(data())`.

15.11 Class template basic_clip_props [io2d.clipprops]

15.11.1 Overview [io2d.clipprops.intro]

- 1 The `basic_clip_props` class template provides general state information that is applicable to all rendering and compositing operations (16.3.2).
- 2 It has a *clip area* of type `optional<interpreted_path>` and a *fill rule* of type `fill_rule`. If the clip area has no value, the clip area is boundless.
- 3 The data are stored in an object of type `typename GraphicsSurfaces::surface_state_props::clip_props_data_type`. It is accessible using the data member functions.

15.11.2 basic_clip_props synopsis [io2d.clipprops.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_clip_props {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surface_state_props::clip_props_data_type;

        // 15.11.3, constructors:
        basic_clip_props() noexcept;
        template <class Allocator>
        explicit basic_clip_props(
            const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
            io2d::fill_rule fr = io2d::fill_rule::winding);
        explicit basic_clip_props(
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            io2d::fill_rule fr = io2d::fill_rule::winding) noexcept;
        explicit basic_clip_props(const basic_bounding_box<graphics_math_type>& r,
            io2d::fill_rule fr = io2d::fill_rule::winding);

        // 15.11.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;
    };
}
```

```

// 15.11.5, modifiers:
void clip();
template <class Allocator>
void clip(const basic_path_builder<GraphicsSurfaces, Allocator>& pb);
void clip(const basic_interpreted_path<GraphicsSurfaces>& ip) noexcept;
void fill_rule(io2d::fill_rule fr) noexcept;

// 15.11.6, observers:
optional<basic_interpreted_path<GraphicsSurfaces>> clip() const noexcept;
io2d::fill_rule fill_rule() const noexcept;
};
}

```

15.11.3 basic_clip_props constructors

[io2d.clipprops.ctor]

```
basic_clip_props() noexcept;
```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_clip_props()`.

3 *Remarks:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>()`. The fill rule is `io2d::fill_rule::winding`.

```
template <class Allocator>
explicit basic_clip_props(const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
    io2d::fill_rule fr = io2d::fill_rule::winding);
```

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

5 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_clip_props(pb, fr)`.

Remarks: The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(pb))`.

7 The fill rule is `fr`.

```
template <class InputIterator>
basic_clip_props(InputIterator first, InputIterator last,
    io2d::fill_rule fr = io2d::fill_rule::winding);
```

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

9 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_clip_props(first, last, fr)`.

Remarks: The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(first, last))`.

11 The fill rule is `fr`.

```
template <class Allocator>
explicit basic_clip_props(
    initializer_list<basic_figure_items<GraphicsSurfaces>::figure_item> il,
    io2d::fill_rule fr = io2d::fill_rule::winding);
```

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

13 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_clip_props(il, fr)`.

Remarks: The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(il))`.

15 The fill rule is `fr`.

```
explicit basic_clip_props(const basic_interpreted_path<GraphicsSurfaces>& ip,
    io2d::fill_rule fr = io2d::fill_rule::winding) noexcept;
```

16 *Effects:* Constructs an object of type `basic_clip_props`.

17 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_clip_props(ip, fr)`.

Remarks: The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>(ip)`.

19 The fill rule is `fr`.

```
explicit basic_clip_props(const basic_bounding_box<graphics_math_type>& r,
    io2d::fill_rule fr = io2d::fill_rule::winding)
```

20 *Effects:* Constructs an object of type `basic_clip_props`.

21 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_clip_props(r, fr)`.

Remarks: The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(r))`;

23 The fill rule is `fr`.

15.11.4 Accessors

[io2d.clipprops.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_clip_props` object's data object (See: [15.11.1](#)).

15.11.5 `basic_clip_props` modifiers

[io2d.clipprops.mod]

```
template <class Allocator>
void clip();
```

1 *Effects:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>()`.

```
void clip(const basic_bounding_box<GraphicsSurfaces>& bb);
```

2 *Effects:* Calls `GraphicsSurfaces::surface_state_props::clip(data(), bb)`;

3 *Remarks:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(bb))`.

```
template <class Allocator>
void clip(const basic_path_builder<GraphicsSurfaces, Allocator>& pb);
```

4 *Effects:* Calls `GraphicsSurfaces::surface_state_props::clip(data(), pb)`;

5 *Remarks:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(pb))`.

```
template <class InputIterator>
void clip(InputIterator first, InputIterator last);
```

6 *Effects:* Calls `GraphicsSurfaces::surface_state_props::clip(data(), first, last)`;

7 *Remarks:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(first, last))`.

```
void clip(
    const initializer_list<typename
    basic_figure_items<GraphicsSurfaces>::figure_item> il);
```

8 *Effects:* Calls `GraphicsSurfaces::surface_state_props::clip(data(), il)`;

9 *Remarks:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(basic_interpreted_path<GraphicsSurfaces>(il))`.

```
void clip(const basic_bounding_box<GraphicsSurfaces>& bb);
```

10 *Effects:* Calls `GraphicsSurfaces::surface_state_props::clip(data(), ip)`;

11 *Remarks:* The clip area is `optional<basic_interpreted_path<GraphicsSurfaces>>(ip)`.

```
void fill_rule(experimental::io2d::fill_rule fr) noexcept;
```

12 *Effects:* Calls `GraphicsSurfaces::surface_state_props::clip_fill_rule(fr)`.

13 *Remarks:* The fill rule is `fr`.

15.11.6 basic_clip_props observers [io2d.clipprops.obs]

```
optional<basic_interpreted_path<GraphicsSurfaces>> clip() const noexcept;
```

1 *Returns:* GraphicsSurfaces::surface_state_props::clip(data()).

2 *Remarks:* The return value is the clip area.

```
io2d::fill_rule fill_rule() const noexcept;
```

3 *Returns:* GraphicsSurfaces::surface_state_props::clip_fill_rule(data()).

4 *Remarks:* The return value is the fill rule.

15.12 Class template basic_stroke_props [io2d.strokeprops]

15.12.1 basic_stroke_props summary [io2d.strokeprops.summary]

1 The `basic_stroke_props` class template provides state information that is applicable to the stroking operation (see: 16.3.2 and 16.3.6).

2 It has a *line width* of type `float`, a *line cap* of type `line_cap`, a *line join* of type `line_join`, a *miter limit* of type `float`, and an *antialiasing algorithm* of type `antialias`.

3 The data are stored in an object of type `typename GraphicsSurfaces::surface_state_props::stroke_props_data_type`. It is accessible using the data member functions.

15.12.2 basic_stroke_props synopsis [io2d.strokeprops.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_stroke_props {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surface_state_props::stroke_props_data_type;

        // 15.12.3, constructors:
        basic_stroke_props() noexcept;
        explicit basic_stroke_props(float w, io2d::line_cap lc = io2d::line_cap::none,
            io2d::line_join lj = io2d::line_join::miter, float ml = 10.0f,
            antialias aa = antialias::good) noexcept;

        // 15.12.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 15.12.5, modifiers:
        void line_width(float w) noexcept;
        void line_cap(io2d::line_cap lc) noexcept;
        void line_join(io2d::line_join lj) noexcept;
        void miter_limit(float ml) noexcept;
        void antialiasing(antialias aa) noexcept;

        // 15.12.6, observers:
        float line_width() const noexcept;
        io2d::line_cap line_cap() const noexcept;
        io2d::line_join line_join() const noexcept;
        float miter_limit() const noexcept;
        antialias antialiasing() const noexcept;
        float max_miter_limit() const noexcept;
    };
}
```

15.12.3 basic_stroke_props constructors [io2d.strokeprops.cons]

```
basic_stroke_props() noexcept;
```

1 *Effects:* Equivalent to: `basic_stroke_props(2.0f)`.

```
explicit basic_stroke_props(float w, io2d::line_cap lc = io2d::line_cap::none,
    io2d::line_join lj = io2d::line_join::miter,
    float ml = 10.0f, antialias aa = antialias::good) noexcept;
```

2 *Requires:* $w > 0.0f$. $ml \geq 10.0f$. $ml \leq \text{max_miter_limit}()$.

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

4 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_stroke_props(w, lc, lj, ml, aa)`.

15.12.4 Accessors

[io2d.strokeprops.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_stroke_props` object's data object (See: [15.12.1](#)).

15.12.5 basic_stroke_props modifiers

[io2d.strokeprops.modifiers]

```
void line_width(float w) noexcept;
```

1 *Requires:* $w \geq 0.0f$.

2 *Effects:* The line width is w .

```
void line_cap(io2d::line_cap lc) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surface_state_props::line_cap(data(), lc)`.

Remarks: The line cap is lc .

```
void line_join(io2d::line_join lj) noexcept;
```

4 *Effects:* Calls `GraphicsSurfaces::surface_state_props::line_join(data(), lj)`.

Remarks: The line join is lj .

```
void miter_limit(float ml) noexcept;
```

5 *Requires:* $ml \geq 1.0f$ and $ml \leq \text{max_miter_limit}$.

6 *Effects:* Calls `GraphicsSurfaces::surface_state_props::miter_limit(data(), ml)`.

Remarks: The miter limit is ml .

```
void antialiasing(antialias aa) noexcept;
```

7 *Effects:* Calls `GraphicsSurfaces::surface_state_props::antialiasing(data(), aa)`.

Remarks: The antialiasing algorithm is aa .

15.12.6 basic_stroke_props observers

[io2d.strokeprops.observers]

```
float line_width() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surface_state_props::line_width(data())`.

2 *Remarks:* The returned value is the line width.

```
io2d::line_cap line_cap() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surface_state_props::line_cap(data())`.

4 *Remarks:* The returned value is the line cap.

```
io2d::line_join line_join() const noexcept;
```

5 *Returns:* `GraphicsSurfaces::surface_state_props::line_join(data())`.

6 *Remarks:* The returned value is the line join.

```
float miter_limit() const noexcept;
```

7 *Returns:* `GraphicsSurfaces::surface_state_props::miter_limit(data())`.

8 *Remarks:* The returned value is the miter limit.

```
void antialiasing(antialias aa) noexcept;
```

9 *Returns:* GraphicsSurfaces::surface_state_props::antialiasing(data()).

10 *Remarks:* The returned value is the antialiasing algorithm.

```
float max_miter_limit() const noexcept;
```

11 *Requires:* This value shall be finite and greater than 10.0f.

12 *Returns:* The implementation-defined maximum value of miter limit.

15.13 Class template basic_fill_props [io2d.fillprops]

15.13.1 Overview [io2d.fillprops.intro]

1 The `basic_fill_props` class template provides state information that is applicable to the filling rendering and compositing operation (16.3.2).

2 It has a *fill rule* of type `fill_rule` and an *antialiasing algorithm* of type `antialias`.

3 The data are stored in an object of type `typename GraphicsSurfaces::surface_state_props::fill_props_data_type`. It is accessible using the `data` member functions.

15.13.2 Synopsis [io2d.fillprops.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_fill_props {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surface_state_props::fill_props_data_type;

        // 15.13.3, constructors:
        basic_fill_props() noexcept;
        explicit basic_fill_props(io2d::fill_rule fr,
            antialias aa = antialias::good) noexcept;

        // 15.13.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 15.13.5, modifiers:
        void fill_rule(io2d::fill_rule fr) noexcept;
        void antialiasing(antialias aa) noexcept;

        // 15.13.6, observers:
        io2d::fill_rule fill_rule() const noexcept;
        antialias antialiasing() const noexcept;
    };
}
```

15.13.3 Constructors [io2d.fillprops.ctor]

```
basic_fill_props() noexcept;
```

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

2 *Postconditions:* `data() == X::surface_state_props::create_fill_props()`.

```
explicit basic_fill_props(io2d::fill_rule fr, antialias aa = antialias::good)
    noexcept;
```

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

4 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_fill_props(fr, aa)`.

15.13.4 Accessors [io2d.fillprops.acc]

```
const data_type& data() const noexcept;
```

```
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_fill_props` object's data object (See: [15.13.1](#)).

15.13.5 Modifiers

[[io2d.fillprops.mod](#)]

```
void fill_rule(io2d::fill_rule fr) noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::surface_state_props::fill_rule(data(), fr)`.

Remarks: The fill rule is `fr`.

```
void antialiasing(antialias aa) noexcept;
```

2 *Effects:* Calls `GraphicsSurfaces::surface_state_props::antialiasing(data(), aa)`.

Remarks: The antialiasing algorithm is `aa`.

15.13.6 Observers

[[io2d.fillprops.obs](#)]

```
io2d::fill_rule fill_rule() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surface_state_props::fill_rule(data())`.

2 *Remarks:* The returned value is the fill rule.

```
antialias antialiasing() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surface_state_props::antialiasing(data())`.

4 *Remarks:* The returned value is the antialiasing algorithm.

15.14 Class template `basic_mask_props`

[[io2d.maskprops](#)]

15.14.1 `basic_mask_props` summary

[[io2d.maskprops.summary](#)]

1 The `basic_mask_props` class template provides state information that is applicable to the mask rendering and compositing operation ([16.3.2](#)).

2 It has a *wrap mode* of type `wrap_mode`, a *filter* of type `filter`, and a *mask matrix* of type `matrix_2d`.

3 The data are stored in an object of type `typename GraphicsSurfaces::surface_state_props::mask_props_data_type`. It is accessible using the data member functions.

15.14.2 `basic_mask_props` synopsis

[[io2d.maskprops.synopsis](#)]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_mask_props {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surface_state_props::mask_props_data_type;

        // 15.14.3, constructor:
        basic_mask_props(io2d::wrap_mode w = io2d::wrap_mode::repeat,
            io2d::filter fi = io2d::filter::good,
            const basic_matrix_2d<graphics_math_type>& m = basic_matrix_2d<graphics_math_type>{})
            noexcept;

        // 15.14.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 15.14.5, modifiers:
        void wrap_mode(io2d::wrap_mode w) noexcept;
        void filter(io2d::filter fi) noexcept;
        void mask_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;
```

```

// 15.14.6, observers:
io2d::wrap_mode wrap_mode() const noexcept;
io2d::filter filter() const noexcept;
basic_matrix_2d<graphics_math_type> mask_matrix() const noexcept;
};
}

```

15.14.3 basic_mask_props constructor [io2d.maskprops.cons]

```

basic_mask_props(io2d::wrap_mode w = io2d::wrap_mode::repeat,
io2d::filter fi = io2d::filter::good,
const basic_matrix_2d<graphics_math_type>& m = basic_matrix_2d<graphics_math_type>{}) noexcept;

```

Requires: `m.is_invertible() == true`.

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

² *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_mask_props(w, fi, m)`.

15.14.4 Accessors [io2d.maskprops.acc]

```

const data_type& data() const noexcept;
data_type& data() noexcept;

```

¹ *Returns:* A reference to the `basic_mask_props` object's data object (See: [15.14.1](#)).

15.14.5 basic_mask_props modifiers [io2d.maskprops.modifiers]

```

void wrap_mode(io2d::wrap_mode w) noexcept;

```

¹ *Effects:* The wrap mode is `w`.

```

void filter(io2d::filter fi) noexcept;

```

² *Effects:* The filter is `fi`.

```

void mask_matrix(const basic_matrix_2d<graphics_math_type>& m) noexcept;

```

³ *Requires:* `m.is_invertible() == true`.

⁴ *Effects:* The mask matrix is `m`.

15.14.6 basic_mask_props observers [io2d.maskprops.observers]

```

io2d::wrap_mode wrap_mode() const noexcept;

```

¹ *Returns:* The wrap mode.

```

io2d::filter filter() const noexcept;

```

² *Returns:* The filter.

```

basic_matrix_2d<graphics_math_type> mask_matrix() const noexcept;

```

³ *Returns:* The mask matrix.

15.15 Class template basic_dashes [io2d.dashes]

15.15.1 basic_dashes class template [io2d.dashes.intro]

¹ The class template `basic_dashes` describes a pattern for determining, in conjunction with other properties, what points on a path are included when a stroking operation is performed.

² It has an *offset* of type `float` and a *pattern* of an *unspecified* type capable of sequentially storing floating-point values.

³ The data are stored in an object of type `typename GraphicsSurfaces::surface_props_data::dashes_props_data_type`. It is accessible using the `data` member function.

15.15.2 Synopsis

[io2d.dashes.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_dashes {
    public:
        using data_type =
            typename GraphicsSurfaces::surface_state_props::dashes_data_type;
    public:
        // 15.15.3, constructors:
        basic_dashes() noexcept;
        template <class InputIterator>
        basic_dashes(float o, InputIterator first, InputIterator last);
        basic_dashes(float o, initializer_list<float> il);

        // 15.15.4, observers:
        const data_type& data() const noexcept;
    };

    // 15.15.5, operators:
    template <class GraphicsSurfaces>
    bool operator==(const basic_dashes<GraphicsSurfaces>& lhs,
        const basic_dashes<GraphicsSurfaces>& rhs) noexcept;
    template <class GraphicsSurfaces>
    bool operator!=(const basic_dashes<GraphicsSurfaces>& lhs,
        const basic_dashes<GraphicsSurfaces>& rhs) noexcept;
}

```

15.15.3 Constructors

[io2d.dashes.cons]

```
basic_dashes() noexcept;
```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_dashes()`.

3 *Remarks:* The offset is `0.0f` and the pattern contains no values.

```

template <class InputIterator>
basic_dashes(float o, InputIterator first, InputIterator last);

```

4 *Requires:* The value type of `InputIterator` is `float`.

5 Each value from `first` through `last - 1` is greater than or equal to `0.0f`.

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

7 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_dashes(o, first, last)`.

8 *Remarks:* The offset is `o` and the pattern is the sequential list of value beginning at `first` and ending at `last - 1`.

```
basic_dashes(float o, initializer_list<float> il);
```

9 *Requires:* Each value in `il` is greater than or equal to `0.0f`.

10 *Effects:* Constructs an object of type `basic_dashes`.

11 *Postconditions:* `data() == GraphicsSurfaces::surface_state_props::create_dashes(o, il)`.

15.15.4 Observers

[io2d.dashes.observers]

```
const data_type& data() const noexcept;
```

1 *Returns:* A reference to the `basic_dashes` object's data object (See [15.15.1](#)).

15.15.5 Operators

[io2d.dashes.ops]

1

```
template <class GraphicsSurfaces>
bool operator==(const basic_dashes<GraphicsSurfaces>& lhs,
  const basic_dashes<GraphicsSurfaces>& rhs) noexcept;
2   Returns: GraphicsSurfaces::surface_state_props::equal(lhs.data(), rhs.data()).
3
template <class GraphicsSurfaces>
bool operator!=(const basic_dashes<GraphicsSurfaces>& lhs,
  const basic_dashes<GraphicsSurfaces>& rhs) noexcept;
4   Returns: GraphicsSurfaces::surface_state_props::not_equal(lhs.data(), rhs.data()).
```


16 Surfaces

[io2d.surfaces]

16.1 Enum class `refresh_style`

[io2d.refreshstyle]

16.1.1 `refresh_style` summary

[io2d.refreshstyle.summary]

- ¹ The `refresh_style` enum class describes when the *draw callback* (Table 45) of a `basic_output_surface` or `basic_unmanaged_output_surface` object shall be called. See Table 39 for the meaning of each `refresh_style` enumerator.

16.1.2 `refresh_style` synopsis

[io2d.refreshstyle.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class refresh_style {
        as_needed,
        as_fast_as_possible,
        fixed
    };
}
```

16.1.3 `refresh_style` enumerators

[io2d.refreshstyle.enumerators]

Table 39 — `refresh_style` value meanings

Enumerator	Meaning
<code>as_needed</code>	The draw callback shall be called when the implementation needs to do so. [<i>Note:</i> The intention of this enumerator is that implementations will call the draw callback as little as possible in order to minimize power usage. Users can call the <code>redraw_required</code> member function of one of the display surface types to make the implementation run the draw callback whenever the user requires. — <i>end note</i>]
<code>as_fast_as_possible</code>	The draw callback shall be called as frequently as possible, subject to any limits of the execution environment.

Table 39 — `refresh_style` value meanings (continued)

Enumerator	Meaning
<code>fixed</code>	<p>The draw callback shall be called as frequently as needed to maintain the <i>desired frame rate</i> (Table 45) as closely as possible. If more time has passed between two successive calls to the draw callback than is required, it shall be called <i>excess time</i> and it shall count towards the <i>required time</i>, which is the time that is required to pass after a call to the draw callback before the next successive call to the draw callback shall be made. If the excess time is greater than the required time, implementations shall call the draw callback and then repeatedly subtract the required time from the excess time until the excess time is less than the required time. If the implementation needs to call the draw callback for some other reason, it shall use that call as the new starting point for maintaining the desired frame rate.</p> <p>[<i>Example</i>: Given a desired frame rate of 20.0f, then as per the above, the implementation would call the draw callback at 50 millisecond intervals or as close thereto as possible. If for some reason the excess time is 51 milliseconds, the implementation would call the draw callback, subtract 50 milliseconds from the excess time, and then would wait 49 milliseconds before calling the draw callback again. If only 15 milliseconds have passed since the draw callback was last called and the implementation needs to call the draw callback again, then the implementation shall call the draw callback immediately and proceed to wait 50 milliseconds before calling the draw callback again. — <i>end example</i>]</p>

16.2 Enum class `image_file_format` [io2d.imagefileformat]

16.2.1 `image_file_format` summary [io2d.imagefileformat.summary]

- ¹ The `image_file_format` enum class specifies the data format that an `image_surface` object is constructed from or saved to. This allows data in a format that is required to be supported to be read or written regardless of its extension.
- ² It also has a value that allows implementations to support additional file formats if it recognizes them.

16.2.2 `image_file_format` synopsis [io2d.imagefileformat.synopsis]

```
namespace std::experimental::io2d::v1 {
    enum class image_file_format {
        unknown,
        png,
        jpeg,
        tiff,
        svg
    };
}
```

16.2.3 `image_file_format` enumerators [io2d.imagefileformat.enumerators]

Table 40 — `image_file_format` enumerator meanings

Enumerator	Meaning
<code>unknown</code>	The format is unknown because it is not an image file format that is required to be supported. It may be known and supported by the implementation.

Table 40 — `image_file_format` enumerator meanings (continued)

Enumerator	Meaning
<code>png</code>	The PNG format.
<code>jpeg</code>	The JPEG format.
<code>tiff</code>	The TIFF format.
<code>svg</code>	The SVG 1.1 Standard format.

16.3 Overview of surface classes [io2d.surface]

16.3.1 Surface class templates description [io2d.surface.intro]

- ¹ There are three *surface class templates*:
 - (1.1) — `basic_image_surface`
 - (1.2) — `basic_output_surface`
 - (1.3) — `basic_unmanaged_output_surface`
- ² For ease of description, an instantiation of a surface class template will be called a *surface*.
- ³ A surface contains visual data and provides an interface for managing and manipulating that visual data.
- ⁴ Surface class templates are `MoveConstructible` and `MoveAssignable`. They are neither `CopyConstructible` nor `CopyAssignable`. [*Note*: On many platforms, especially those that use specialized hardware to accelerate various graphics operations, copying a surface is highly detrimental to performance and is rarely desired. The `copy_surface` function (19.2) exists for those situations where a copy is desired. — *end note*]
- ⁵ The surface class templates manipulate visual data through rendering and compositing operations.
- ⁶ The rendering and compositing operations 16.3.2 are described in terms of operating on each integral point of the visual data of a surface. The reason for that is to support the discrete nature of raster graphics data. Operating on each integral point of the surface is the coarsest granularity allowed. Implementations may perform rendering and compositing operations at a finer granularity than that of each integral point. [*Note*: Vector graphics data, being continuous, has the finest granularity possible since it resolves at the limits imposed by the precision of the floating-point types used to determine its visual data at any particular point. — *end note*]

16.3.2 Rendering and compositing [io2d.surface.rendering]

16.3.2.1 Operations [io2d.surface.rendering.ops]

- ¹ The surface classes provide five fundamental rendering and compositing operations:

Table 41 — surface rendering and compositing operations

Operation	Function(s)
Painting	<code>paint</code>
Filling	<code>fill</code>
Stroking	<code>stroke</code>
Masking	<code>mask</code>
Text Rendering	<code>draw_text</code>

- ² All compositing operations shall happen as-if in a linear color space, regardless of the color space of the visual data that is involved.
- ³ [*Note*: While a color space such as sRGB helps produce expected, consistent results when visual data are viewed by people, compositing operations only produce expected results when the valid values for the color channel and alpha channel data in the visual data involved are uniformly (i.e. linearly) spaced. — *end note*]

16.3.2.2 Rendering and compositing brushes [io2d.surface.rendering.brushes]

- ¹ All rendering and compositing operations use a *source brush* of type `basic_brush`.
- ² The masking operation uses a *mask brush* of type `basic_brush`.

16.3.2.3 Rendering and composing source path [io2d.surface.rendering.sourcepath]

- ¹ In addition to brushes (16.3.2.2), the Stroke and Fill rendering and composing operations use a *source path*. The source path is either a `basic_path_builder<Allocator>` object or a `basic_interpreted_path` object. If it is a `basic_path_builder<Allocator>` object, it is interpreted (13.3.16) before it is used as the source path.
- ² In addition to brushes, the Text Rendering rendering and composing operation uses a *source text* and a *font*. The source text is a `string` object containing text in the UTF-8 text format. The font is a `basic_font` object.

16.3.2.4 Common state data [io2d.surface.rendering.commonstate]

- ¹ All rendering and composing operations use the following state data:

Table 42 — surface rendering and composing common state data

Name	Type
Brush properties	<code>basic_brush_props</code>
Surface properties	<code>basic_render_props</code>
Clip properties	<code>basic_clip_props</code>

16.3.2.5 Specific state data [io2d.surface.rendering.specificstate]

- ¹ In addition to the common state data (16.3.2.4), certain rendering and composing operations use state data that is specific to each of them:

Table 43 — surface rendering and composing specific state data

Operation	Name	Type
Stroking	Stroke properties	<code>basic_stroke_props</code>
Stroking	Dashes	<code>basic_dashes</code>
Filling	Fill properties	<code>basic_fill_props</code>
Masking	Mask properties	<code>mask_props</code>
Text Rendering	Text properties	<code>basic_text_props</code>

16.3.2.6 State data default values [io2d.surface.rendering.statedefaults]

- ¹ For all rendering and composing operations, the state data objects named above are provided using `optional<T>` class template arguments.
- ² If there is no contained value for a state data object, it is interpreted as-if the `optional<T>` argument contained a default constructed object of the relevant state data object.

16.3.3 Standard coordinate spaces [io2d.surface.coordinatespaces]

- ¹ There are four standard coordinate spaces relevant to the rendering and composing operations (16.3.2):
 - (1.1) — the brush coordinate space;
 - (1.2) — the mask coordinate space;
 - (1.3) — the user coordinate space; and
 - (1.4) — the surface coordinate space.
- ² The *brush coordinate space* is the standard coordinate space of the source brush (16.3.2.2). Its transformation matrix is the brush properties' brush matrix (15.10.1).
- ³ The *mask coordinate space* is the standard coordinate space of the mask brush (16.3.2.2). Its transformation matrix is the mask properties' mask matrix (15.14.1).
- ⁴ The *user coordinate space* is the standard coordinate space of `basic_interpreted_path` objects. Its transformation matrix is a default-constructed `basic_matrix_2d`.
- ⁵ The *surface coordinate space* is the standard coordinate space of the surface object's visual data. Its transformation matrix is the surface properties' surface matrix (15.9.1).

- ⁶ Given a point `pt`, a brush coordinate space transformation matrix `bcsm`, a mask coordinate space transformation matrix `mcsm`, a user coordinate space transformation matrix `ucsm`, and a surface coordinate space transformation matrix `scsm`, the following table describes how to transform it from each of these standard coordinate spaces to the other standard coordinate spaces:

Table 44 — Point transformations

From	To	Transform
brush coordinate space	mask coordinate space	<code>mcsm.transform_pt(bcsm.invert().transform_pt(pt))</code> .
brush coordinate space	user coordinate space	<code>bcsm.invert().transform_pt(pt)</code> .
brush coordinate space	surface coordinate space	<code>scsm.transform_pt(bcsm.invert().transform_pt(pt))</code> .
user coordinate space	brush coordinate space	<code>bcsm.transform_pt(pt)</code> .
user coordinate space	mask coordinate space	<code>mcsm.transform_pt(pt)</code> .
user coordinate space	surface coordinate space	<code>scsm.transform_pt(pt)</code> .
surface coordinate space	brush coordinate space	<code>bcsm.transform_pt(scsm.invert().transform_pt(pt))</code> .
surface coordinate space	mask coordinate space	<code>mcsm.transform_pt(scsm.invert().transform_pt(pt))</code> .
surface coordinate space	user coordinate space	<code>scsm.invert().transform_pt(pt)</code> .

16.3.4 surface painting

[io2d.surface.painting]

- ¹ When a painting operation is initiated on a surface, the implementation shall produce results as-if the following steps were performed:
1. For each integral point `sp` of the surface’s visual data, determine if `sp` is within the clip area (15.11.1); if so, proceed with the remaining steps.
 2. Transform `sp` from the surface coordinate space (16.3.3) to the brush coordinate space (Table 44), resulting in point `bp`.
 3. Sample from point `bp` of the source brush (16.3.2.2), combine the resulting visual data with the visual data at point `sp` in the surface’s visual data in the manner specified by the surface’s current *compositing operator* (15.9.1), and modify the visual data of the surface at point `sp` to reflect the result produced by application of the compositing operator.

16.3.5 surface filling

[io2d.surface.filling]

- ¹ When a filling operation is initiated on a surface, the implementation shall produce results as-if the following steps were performed:
1. For each integral point `sp` of the surface’s visual data, determine if `sp` is within the *clip area* (15.11.1); if so, proceed with the remaining steps.
 2. Transform `sp` from the surface coordinate space (16.3.3) to the user coordinate space (Table 44), resulting in point `up`.
 3. Using the source path (16.3.2.3), the fill rule (15.13.1), and the antialiasing algorithm (15.13.1), determine whether `up` shall be filled; if so, proceed with the remaining steps.
 4. Transform `up` from the user coordinate space to the brush coordinate space (16.3.3 and Table 44), resulting in point `bp`.
 5. Sample from point `bp` of the source brush (16.3.2.2), combine the resulting visual data with the visual data at point `sp` in the surface’s visual data in the manner specified by the surface’s current compositing operator (15.9.1), apply the antialiasing algorithm if applicable, and modify the surface’s visual data at point `sp` to reflect the result produced by application of the compositing operator and antialiasing algorithm.

16.3.6 surface stroking**[io2d.surface.stroking]**

- ¹ When a stroking operation is initiated on a surface, it is carried out for each figure in the source path (16.3.2).
- ² The following rules shall apply when a stroking operation is carried out on a figure:
 1. No part of the surface’s visual data that is outside of the clip area shall be modified.
 2. The antialiasing algorithm (15.12.1) shall be used where applicable to determine whether a point along the path is rendered and if so what its resulting visual data is.
 3. If the figure is a closed figure, then the point where the end point of its final segment meets the start point of the initial segment shall be rendered as specified by the *line join* value (see: 15.12.1 and 16.3.2.5); otherwise the start point of the initial segment and end point of the final 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 (Table 43) has its default value or if its **vector<float>** member is empty, the segments shall be rendered as a continuous path.
 5. If the dash pattern’s **vector<float>** member contains only one value, that value shall be used to define a repeating pattern in which the path is shown then hidden. The ends of each shown portion of the path shall be rendered as specified by the line cap value.
 6. If the dash pattern’s **vector<float>** member contains two or more values, the values shall be used to define a pattern in which the figure is alternatively rendered then not rendered for the length specified by the value. The ends of each rendered portion of the figure shall be rendered as specified by the line cap value. If the dash pattern’s **float** member, which specifies an offset value, is not **0.0f**, the meaning of its value is implementation-defined. If a rendered portion of the figure overlaps a not rendered portion of the figure, the rendered portion shall be rendered.
- ³ When a stroking operation is carried out on a figure, the width of each rendered portion shall be the *line width* (see: 15.12.1 and 16.3.2.5). Ideally this means that the diameter of the stroke at each rendered point should be equal to the line width. However, because there are an infinite number of points along each rendered portion, implementations may choose an unspecified method of determining minimum distances between points along each rendered portion and the diameter of the stroke between those points shall be the same. [*Note: This concept is sometimes referred to as a tolerance. It allows for a balance between precision and performance, especially in situations where the end result is in a non-exact format such as raster graphics data. — end note*]
- ⁴ After all figures in the path have been rendered but before the rendered result is composed to the surface’s visual data, the rendered result shall be transformed from the user coordinate space (16.3.3) to the surface coordinate space (16.3.3).

16.3.7 surface masking**[io2d.surface.masking]**

- ¹ When a masking operation is initiated on a surface, the implementation shall produce results as-if the following steps were performed:
 1. For each integral point *sp* of the surface’s visual data, determine if *sp* is within the clip area (15.11.1); if so, proceed with the remaining steps.
 2. Transform *sp* from the surface coordinate space (16.3.3) to the mask coordinate space (Table 44), resulting in point *mp*.
 3. Sample the alpha channel from point *mp* of the mask brush and store the result in *mac*; if the visual data format of the mask brush does not have an alpha channel, the value of *mac* shall always be 1.0.
 4. Transform *sp* from the surface coordinate space to the brush coordinate space, resulting in point *bp*.
 5. Sample from point *bp* of the source brush (16.3.2.2), combine the resulting visual data with the surface’s visual data at point *sp* in the manner specified by the surface’s current compositing operator (15.9.1), multiply each channel of the result produced by application of the compositing operator by *map* if the visual data format of the surface’s visual data is a premultiplied format and if not then just multiply the alpha channel of the result by *map*, and modify the surface’s visual data at point *sp* to reflect the multiplied result.

16.3.8 surface text rendering

[io2d.surface.textrendering]

- ¹ [*Note:* The following uses terminology and other information contained in ISO/IEC 10646 and ISO/IEC 14496-22, both of which are listed in the normative references of this Technical Specification . — *end note*]
- ² Text rendering is a complex subject. The specifics of how it is performed are described in ISO/IEC 10646 and ISO/IEC 14496-22. The following is an informative overview of the process. The normative process is described in those two standards, except where otherwise noted.
1. This item is normative. The source text is transformed from UTF-8 to UCS characters.
 2. This item is normative. The UCS characters are transformed into glyphs using the cmap table of the font. Where any characters do not map to glyphs, if the `basic_font` object's `merging` value is `true`, a font with a similar style (which should be determined using the OS/2 table's `sFamilyClass` value or in an unspecified manner if that matching method fails to produce a result) that can make that mapping is selected and used for those characters. If no such font exists or if the `basic_font` object's `merging` value is `false`, the characters that do not map to glyphs are mapped to glyph index 0 as noted in the description of the cmap table.
 3. That set of glyphs is then modified using transformations specified in various tables in the font, some of which are optional and thus may not be present, e.g. the GSUB table. Depending on the glyphs that are present in the set and the presence or absence of certain tables, it is possible that no modification will occur.
 4. The layout of the glyphs for purposes of rasterization is calculated using metrics and adjustments specified in various tables in the font, some of which are optional, e.g. the BASE and GPOS tables.
 5. This item is normative. When the text properties' location is a `basic_point_2d` object, the glyphs are rendered in a single line without regard to justification. When the location is a `basic_bounding_box` object, the glyphs are rendered in one or more lines, which are contained within the confines of the `basic_bounding_box` object.
 6. If applicable, potential line break locations are determined.
 7. This item is normative. The layout is modified as specified by the property values contained within the text properties.
 8. If multiple lines are possible, the layout is modified to ensure that the rendered glyphs will fit within the specified bounds, adding line breaks where necessary and applying justification to each line.
 9. This item is normative. Where the glyph data are contours or otherwise are composed of commands that resemble the figure items described in this Technical Specification , the glyphs shall be rendered and composed as-if this was a stroking operation, using the glyph data as the source path after applying any mathematical transformations necessary to convert the glyph data as defined in ISO/IEC 14496-22 into figure items as defined in this Technical Specification . Where the glyph data are raster graphics data, the glyphs shall be rendered and composed as-if this was a masking operation, with the glyph data serving as the mask brush and a `basic_mask_props` object default constructed with its wrap mode changed to `wrap_mode:none` serving as the mask properties. Regardless of how the glyph data are rendered and composed, the glyph data shall be transformed from its design space to surface coordinate space as specified in ISO/IEC 14496-22 suitably adjusted to conform to the layout. [*Note:* The property values contained in the text properties have already been applied to the layout such that things like its font size and size units will be taken into account when transformation of glyph data from its design space to surface coordinate space occurs. — *end note*]

16.3.9 output surface miscellaneous behavior

[io2d.outputsurface.misc]

- ¹ What constitutes an *output device* is implementation-defined, with the sole constraint being that an output device must allow the user to see the dynamically-updated contents of the display buffer. [*Example:* An output device might be a window in a windowing system environment or the usable screen area of a smart phone or tablet. — *end example*]
- ² Implementations may allow more than one `basic_output_surface` object, `basic_unmanaged_output_surface` object, or a combination thereof to exist and be displayed co-synchronously. [*Note:* In windowing environments, implementations would likely support multiple objects of these types. In contrast, on a smart phone it is unlikely that an implementation would support multiple objects of these types due to environmental and platform constraints. — *end note*]

- ³ It is not required that implementations support the existence of any `basic_unmanaged_output_surface` objects. See Table 16.
- ⁴ All functions that perform rendering and composing operations operate on the back buffer. The data is subsequently transferred to the display buffer as specified by the output surfaces.

16.3.10 output surface state

[io2d.outputsurface.state]

- ¹ Table 45 specifies the name, type, function, and default value for each item of a display surface's observable state.

Table 45 — Output surface observable state

Name	Type	Function	Default value
<i>Letterbox brush</i>	<code>brush</code>	This is the brush that shall be used as specified by <code>scaling::letterbox</code> (Table 38)	<code>brush{ { rgba_color::black } }</code>
<i>Letterbox brush props</i>	<code>brush_props</code>	This is the brush properties for the letterbox brush	<code>brush_props{ }</code>
<i>Scaling type</i>	<code>scaling</code>	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	<code>scaling::letterbox</code>
<i>Draw width</i>	<code>int</code>	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	<i>N/A</i> [<i>Note: It is impossible to create an output surface object without providing a preferred draw width value; as such a default value cannot exist. — end note</i>]

Table 45 — Output surface observable state (continued)

Name	Type	Function	Default value
<i>Draw height</i>	int	The height in pixels of the back buffer. The minimum value is 1. The maximum value is unspecified. Because users can only request a preferred value for the draw height when setting and altering it, the maximum value may be a run-time determined value. If the preferred draw height exceeds the maximum value, then if a preferred draw width has also been supplied then implementations should provide a back buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred draw width and the preferred draw height otherwise implementations should provide a back buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the current draw width and the preferred draw height	<i>N/A</i> [<i>Note</i> : It is impossible to create an output surface object without providing a preferred draw height value; as such a default value cannot exist. — <i>end note</i>]
<i>Draw format</i>	format	The pixel format of the back buffer. When an output surface object is created, a preferred pixel format value is provided. If the implementation does not support the preferred pixel format value as the value of draw format, the resulting value of draw format is implementation-defined	<i>N/A</i> [<i>Note</i> : It is impossible to create an output surface object without providing a preferred draw format value; as such a default value cannot exist. — <i>end note</i>]

Table 45 — Output surface observable state (continued)

Name	Type	Function	Default value
<i>Display width</i>	int	The width in pixels of the display buffer. The minimum value is unspecified . The maximum value is unspecified . Because users can only request a preferred value for the display width when setting and altering it, both the minimum value and the maximum value may be run-time determined values. If the preferred display width is not within the range between the minimum value and the maximum value, inclusive, then if a preferred display height has also been supplied then implementations should provide a display buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred display width and the preferred display height otherwise implementations should provide a display buffer with the largest dimensions possible that maintain as nearly as possible the aspect ratio between the preferred display width and the current display height	<i>N/A</i> [<i>Note</i> : It is impossible to create an output 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. — <i>end note</i>]

Table 45 — Output surface observable state (continued)

Name	Type	Function	Default value
<i>Display height</i>	<code>int</code>	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	<i>N/A</i> [<i>Note</i> : It is impossible to create an output 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. — <i>end note</i>]
<i>Auto clear</i>	<code>bool</code>	If <code>true</code> the implementation shall call <code>clear</code> , which shall clear the back buffer, immediately before it executes the draw callback	<code>false</code>
<i>Refresh style</i>	<code>refresh_style</code>	The <code>refresh_style</code> value that determines when the draw callback shall be called while <code>basic_output_surface<T>::begin_show</code> is being executed	<code>refresh_style::as_fast_as_possible</code>
<i>Desired frame rate</i>	<code>float</code>	This value is the number of times the draw callback shall be called per second while <code>basic_output_surface<T>::begin_show</code> is being executed when the value of refresh style is <code>refresh_style::fixed</code> , subject to the additional requirements documented in the meaning of <code>refresh_style::fixed</code> (See: Table 39).	<code>30.0f</code>

16.4 Class `basic_image_surface` [io2d.imagesurface]

16.4.1 `basic_image_surface` summary [io2d.imagesurface.summary]

- ¹ The class `basic_image_surface` provides an interface to raster graphics data.
- ² It has a *pixel format* of type `format`, a *width* of type `int`, and a *height* of type `int`.
- ³ The data are stored in an object of type `typename GraphicsSurfaces::surfaces::image_surface_data_type`. It is accessible using the data member functions.
- ⁴ [*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 `basic_image_surface` class template as being a form of *render target*. This is intentional, though this Technical Specification does not formally define or use that term to avoid any minor ambiguities and differences in its meaning between the various graphics technologies that do use the term *render target*. — *end note*]

16.4.2 `basic_image_surface` synopsis [io2d.imagesurface.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_image_surface {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::image_surface_data_type;

        // 16.4.3, construct/copy/move/destroy:
        basic_image_surface(io2d::format fmt, int width, int height);
        basic_image_surface(filesystem::path f, io2d::image_file_format iff, io2d::format fmt);
        basic_image_surface(filesystem::path f, io2d::image_file_format iff, io2d::format fmt,
            error_code& ec) noexcept;
        basic_image_surface(basic_image_surface&&) noexcept;
        basic_image_surface& operator=(basic_image_surface&&) noexcept;

        // 16.4.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 16.4.5, members:
        void save(filesystem::path p, image_file_format i);
        void save(filesystem::path p, image_file_format i, error_code& ec) noexcept;

        // 16.4.6, static members:
        static basic_display_point<graphics_math_type> max_dimensions() noexcept;

        // 16.4.7, observers:
        io2d::format format() const noexcept;
        basic_display_point<graphics_math_type> dimensions() const noexcept;

        // 16.4.8, modifiers:
        void clear();
        void paint(const basic_brush<GraphicsSurfaces>& b,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
        template <class Allocator>
        void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
            const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
        void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
```

```

    const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
template <class Allocator>
void fill(const basic_brush<GraphicsSurfaces>& b,
    const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void fill(const basic_brush<GraphicsSurfaces>& b,
    const basic_interpreted_path<GraphicsSurfaces>& ip,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void mask(const basic_brush<GraphicsSurfaces>& b,
    const basic_brush<GraphicsSurfaces>& mb,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_mask_props<GraphicsSurfaces>>& mp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void draw_text(const basic_point_2d<graphics_math_type>& pt,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, const string& text,
    const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
    const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void draw_text(const basic_bounding_box<graphics_math_type>& bb,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, const string& text,
    const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
    const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
    future<void> command_list(const basic_command_list<GraphicsSurfaces>& cl);
};
}

```

16.4.3 basic_image_surface constructors and assignment operators [io2d.imagesurface.cons]

```
basic_image_surface(io2d::format fmt, int w, int h);
```

1 *Requires:* w is greater than 0 and not greater than basic_image_surface::max_width().

2 h is greater than 0 and not greater than basic_image_surface::max_height().

3 fmt is not io2d::format::invalid.

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

5 The pixel format is fmt, the width is w, and the height is h.

6 *Postconditions:* data() == GraphicsSurfaces::surfaces::create_image_surface(fmt, w, h).

```
basic_image_surface(filesystem::path f, io2d::image_file_format i, io2d::format fmt);
```

```
basic_image_surface(filesystem::path f, io2d::image_file_format i, io2d::format fmt,
    error_code& ec) noexcept;
```

7 *Requires:* f is a file and its contents are data in a supported format (see: 16.2).

8 fmt is not io2d::format::invalid.

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

10 *Postconditions:* If called without an `error_code&` argument `data() == GraphicsSurfaces::surfaces::create_image_surface(f, i, fmt)`, otherwise `data() == GraphicsSurfaces::surfaces::create_image_surface(f, i, fmt, ec)`.

11 *Remarks:* The raster graphics data is the result of processing `f` into uncompressed raster graphics in the manner specified by the standard that describes how to transform the contents of data contained in `f` into raster graphics data and then transforming that transformed raster graphics data into the format specified by `fmt`.

12 The data of `f` is processed into uncompressed raster graphics data as specified by the value of `i`.

13 If `i` is `image_file_format::unknown`, implementations may attempt to process the data of `f` into uncompressed raster graphics data. The manner in which it does so is unspecified. If no uncompressed raster graphics data is produced, the error specified below occurs.

14 [*Note:* The intent of `image_file_format::unknown` is to allow implementations to support image file formats that are not required to be supported. — *end note*]

15 If the width of the uncompressed raster graphics data would be less than 1 or greater than `basic_image_surface::max_width()` or if the height of the uncompressed raster graphics data would be less than 1 or greater than `basic_image_surface::max_height()`, the error specified below occurs.

16 The resulting uncompressed raster graphics data is then transformed into the data format specified by `fmt`. If the format specified by `fmt` only contains an alpha channel, the values of the color channels, if any, of the surface's visual data are unspecified. If the format specified by `fmt` only contains color channels and the resulting uncompressed raster graphics data is in a premultiplied format, then the value of each color channel for each pixel is be divided by the value of the alpha channel for that pixel. The visual data is then set as the visual data of the surface.

17 The width is the width of the uncompressed raster graphics data. The height is the height of the uncompressed raster graphics data.

18 *Throws:* As specified in Error reporting (Clause 4).

19 *Error conditions:* Any error that could result from trying to access `f`, open `f` for reading, or reading data from `f`.

20 `errc::not_supported` if `image_file_format::unknown` is passed as an argument and the implementation is unable to determine the file format or does not support saving in the image file format it determined.

21 `errc::invalid_argument` if `fmt` is `io2d::format::invalid`.

22 `errc::argument_out_of_domain` if the width would be less than 1, the width would be greater than `basic_image_surface::max_width()`, the height would be less than 1, or the height would be greater than `basic_image_surface::max_height()`.

16.4.4 Accessors

[`io2d.imagesurface.acc`]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_image_surface` object's data object (See: 16.4.1).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

16.4.5 `basic_image_surface` members

[`io2d.imagesurface.members`]

```
void save(filesystem::path p, image_file_format i);
void save(filesystem::path p, image_file_format i, error_code& ec) noexcept;
```

1 *Requires:* `p` shall be a valid path to a file. The file need not exist provided that the other components of the path are valid.

2 If the file exists, it shall be writable. If the file does not exist, it shall be possible to create the file at the specified path and then the created file shall be writable.

3 *Effects:* If called without an `error_code&` argument `GraphicsSurfaces::surfaces::save(p, i)`, otherwise `GraphicsSurfaces::surfaces::save(p, i, ec)`.

4 *Remarks:* Any pending rendering and compositing operations (16.3.2) are performed before the surface's visual data is written to `p`.

5 The surface's visual data is written to `p` in the data format specified by `i`.

6 If `i` is `image_file_format::unknown`, it is implementation-defined whether the surface is saved in the image file format, if any, that the implementation associates with `p.extension()` provided that `p.has_extension() == true`. If `p.has_extension() == false`, the implementation does not associate an image file format with `p.extension()`, or the implementation does not support saving in that image file format, the error specified below occurs.

7 *Throws:* As specified in Error reporting (Clause 4).

8 *Error conditions:* Any error that could result from trying to create `f`, access `f`, or write data to `f`.

9 `errc::not_supported` if `image_file_format::unknown` is passed as an argument and the implementation is unable to determine the file format or does not support saving in the image file format it determined.

16.4.6 `basic_image_surface` static members [io2d.imagesurface.staticmembers]

```
static basic_display_point<graphics_math_type> max_dimensions() noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::max_dimensions()`.

2 *Remarks:* The maximum height and width for a `basic_image_surface` object.

16.4.7 `basic_image_surface` observers [io2d.imagesurface.observers]

```
io2d::format format() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::format(data())`.

2 *Remarks:* The pixel format.

```
basic_display_point<graphics_math_type> dimensions() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surfaces::dimensions(data())`.

4 *Remarks:* The pixel dimensions.

16.4.8 `basic_image_surface` modifiers [io2d.imagesurface.modifiers]

```
void clear();
```

1 *Effects:* Equivalent to `paint(basic_brush<GraphicsSurfaces>(rgba_color::white), nullptr, basic_render_props<GraphicsSurfaces>(nearest, basic_matrix_2d<typename GraphicsSurfaces::graphics_math_type>{}, compositing_op::clear))`;

```
void paint(const basic_brush<GraphicsSurfaces>& b,
const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullptr,
const optional<basic_render_props<GraphicsSurfaces>>& rp = nullptr,
const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullptr);
```

2 *Effects:* Calls `GraphicsSurfaces::surfaces::paint(data(), b, (bp == nullptr ? basic_brush_props<GraphicsSurfaces>() : bp.value()), (rp == nullptr ? basic_render_props<GraphicsSurfaces>() : rp.value()), (cl == nullptr ? basic_clip_props<GraphicsSurfaces>() : cl.value()))`

3 *Remarks:* Performs the painting rendering and compositing operation as specified by 16.3.4.

4 The meanings of the parameters are specified by 16.3.2.

5 *Throws:* As specified in Error reporting (Clause 4).

6 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
```

```
void stroke(const basic_brush<GraphicsSurfaces>& b,
const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullptr,
const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullptr,
const optional<basic_dashes<GraphicsSurfaces>>& d = nullptr,
const optional<basic_render_props<GraphicsSurfaces>>& rp = nullptr,
```

```
const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

7 *Effects:* Calls `GraphicsSurfaces::surfaces::stroke(data(), b, basic_interpreted_path<GraphicsSurfaces>(bp == nullopt ? basic_brush_props<GraphicsSurfaces>() : bp.value()), (sp == nullopt ? basic_stroke_props<GraphicsSurfaces>() : sp.value()), (d == nullopt ? basic_dashes<GraphicsSurfaces>() : d.value()), (rp == nullopt ? basic_render_props<GraphicsSurfaces>() : rp.value()), (cl == nullopt ? basic_clip_props<GraphicsSurfaces>() : cl.value()))`.

8 *Remarks:* Performs the stroking rendering and compositing operation as specified by 16.3.6.

9 The meanings of the parameters are specified by 16.3.2.

10 *Throws:* As specified in Error reporting (Clause 4).

11 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
            const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

12 *Effects:* Calls `GraphicsSurfaces::surfaces::stroke(data(), b, ip, (bp == nullopt ? basic_brush_props<GraphicsSurfaces>() : bp.value()), (sp == nullopt ? basic_stroke_props<GraphicsSurfaces>() : sp.value()), (d == nullopt ? basic_dashes<GraphicsSurfaces>() : d.value()), (rp == nullopt ? basic_render_props<GraphicsSurfaces>() : rp.value()), (cl == nullopt ? basic_clip_props<GraphicsSurfaces>() : cl.value()))`.

13 *Remarks:* Performs the stroking rendering and compositing operation as specified by 16.3.6.

14 The meanings of the parameters are specified by 16.3.2.

15 *Throws:* As specified in Error reporting (Clause 4).

16 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
void fill(const basic_brush<GraphicsSurfaces>& b,
          const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
          const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
          const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
          const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
          const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

17 *Effects:* Calls `GraphicsSurfaces::surfaces::fill(data(), b, basic_interpreted_path<GraphicsSurfaces>(pb), (bp == nullopt ? basic_brush_props<GraphicsSurfaces>() : bp.value()), (fp == nullopt ? basic_fill_props<GraphicsSurfaces>() : fp.value()), (rp == nullopt ? basic_render_props<GraphicsSurfaces>() : rp.value()), (cl == nullopt ? basic_clip_props<GraphicsSurfaces>() : cl.value()))`.

18 *Remarks:* Performs the filling rendering and compositing operation as specified by 16.3.5.

19 The meanings of the parameters are specified by 16.3.2.

20 *Throws:* As specified in Error reporting (Clause 4).

21 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void fill(const basic_brush<GraphicsSurfaces>& b,
          const basic_interpreted_path<GraphicsSurfaces>& ip,
          const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
          const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
          const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
          const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

22 *Effects:* Calls `GraphicsSurfaces::surfaces::fill(data(), b, ip, (bp == nullopt ? basic_brush_props<GraphicsSurfaces>() : bp.value()), (fp == nullopt ? basic_fill_props<GraphicsSurfaces>() : fp.value()), (rp == nullopt ? basic_render_props<GraphicsSurfaces>() : rp.value()), (cl == nullopt ? basic_clip_props<GraphicsSurfaces>() : cl.value()))`.

23 *Remarks:* Performs the filling rendering and compositing operation as specified by 16.3.5.

24 The meanings of the parameters are specified by 16.3.2.

25 *Throws:* As specified in Error reporting (Clause 4).

26 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void mask(const basic_brush<GraphicsSurfaces>& b,
         const basic_brush<GraphicsSurfaces>& mb,
         const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
         const optional<basic_mask_props<GraphicsSurfaces>>& mp = nullopt,
         const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
         const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

27 *Effects:* Calls `GraphicsSurfaces::surfaces::mask(data(), b, mb, (bp == nullopt ? basic_brush_props<GraphicsSurfaces>() : bp.value()), (mp == nullopt ? basic_mask_props<GraphicsSurfaces>() : mp.value()), (rp == nullopt ? basic_render_props<GraphicsSurfaces>() : rp.value()), (cl == nullopt ? basic_clip_props<GraphicsSurfaces>() : cl.value()))`.

28 *Remarks:* Performs the masking rendering and compositing operation as specified by 16.3.7.

29 The meanings of the parameters are specified by 16.3.2.

30 *Throws:* As specified in Error reporting (Clause 4).

31 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void draw_text(const basic_point_2d<graphics_math_type>& pt,
              const basic_brush<GraphicsSurfaces>& b,
              const basic_font<GraphicsSurfaces>& font, const string& text,
              const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
              const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
              const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
              const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
              const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
              const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

32 *Effects:* Calls `GraphicsSurfaces::surfaces::draw_text(data(), pt, b, font, text, tp.value_or(basic_text_props<GraphicsSurfaces>()), bp.value_or(basic_brush_props<GraphicsSurfaces>()), sp.value_or(basic_stroke_props<GraphicsSurfaces>()), d.value_or(basic_dashes<GraphicsSurfaces>()), rp.value_or(basic_render_props<GraphicsSurfaces>()), cl.value_or(basic_clip_props<GraphicsSurfaces>()))`.

33 *Remarks:* Performs the text rendering rendering and compositing operation as specified by 16.3.8.

34 The meanings of the parameters are specified by 16.3.2.

35 *Throws:* As specified in Error reporting (Clause 4).

36 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void draw_text(const basic_bounding_box<graphics_math_type>& bb,
              const basic_brush<GraphicsSurfaces>& b,
              const basic_font<GraphicsSurfaces>& font, const string& text,
              const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
              const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
              const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
              const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
              const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
              const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

37 *Effects:* Calls `GraphicsSurfaces::surfaces::draw_text(data(), bb, b, font, text, tp.value_or(basic_text_props<GraphicsSurfaces>()), bp.value_or(basic_brush_props<GraphicsSurfaces>()), sp.value_or(basic_stroke_props<GraphicsSurfaces>()), d.value_or(basic_dashes<GraphicsSurfaces>()), rp.value_or(basic_render_props<GraphicsSurfaces>()), cl.value_or(basic_clip_props<GraphicsSurfaces>()))`.

38 *Remarks:* Performs the text rendering rendering and compositing operation as specified by 16.3.8.

39 The meanings of the parameters are specified by 16.3.2.

40 *Throws:* As specified in Error reporting (Clause 4).

41 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
future<void> command_list(const basic_command_list<GraphicsSurfaces>& cl);
```

42 *Effects:* Calls `GraphicsSurfaces::surfaces::command_list(data(), cl)`.

43 *Returns:* A `future<void>` object to inform the user when the command list has completed.

44 *Remarks:* Submits a command list to be processed by the surface. The command list may run on a separate thread. Users shall be responsible for preventing data races.

45 [*Note:* The ability of the implementation to run command lists on separate threads provides a number of optimization opportunities. As a byproduct, it introduces the potential for data races.

46 Submitting a command list to a `basic_image_surface` object and then calling one of its other functions, especially one of the rendering and compositing operation functions, before the command list has finished execution is highly likely to introduce data races. Attempting to use that object before the command list has finished execution will, at best, produce erroneous results.

47 Some intended uses for command lists are to allow advanced graphics users who need high performance for their applications to run graphics operations in parallel, to allow users to pre-record various sets of operations and run them on an as-needed basis, and to provide a mechanism where users can be sure that the graphics operations they perform will be batched such that the function is guaranteed to return instantly and allow other work that would not introduce data races to proceed in parallel.

48 As such, it is recommended that users choose to use either command lists or the "direct" API (where surface member functions that perform rendering and compositing operations, copying image surfaces, saving image data, etc.).

49 This is not to suggest that the direct API is simplistic and only meant for beginners. For light graphics loads it is easier to use since it does not introduce potentials for race conditions, etc. Further, the direct API can be implemented such that its graphics operations are batched and only run when it is efficient to run them or when when observable behavior requirements force their execution, which will allow them to have reasonably good performance. — *end note*]

16.5 Class `basic_output_surface` [io2d.outputsurface]

16.5.1 `basic_output_surface` summary [io2d.outputsurface.summary]

- 1 A `basic_output_surface` object represents a simple way to display 2D graphics to a user. The mechanisms required to ensure that all environment-specific requirements, such as providing an event loop handler, are provided by the object.
- 2 The user just needs to set a draw callback or a command list (or both) and then call the `begin_show` member function.
- 3 When the user wishes to end the display of 2D graphics, the user calls the `end_show` member function either from the draw callback or from a `run_function` object contained within the command list.
- 4 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::output_surface_data_type`. It is accessible using the `data` member functions.

16.5.2 `basic_output_surface` synopsis [io2d.outputsurface.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_output_surface {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;

        // 16.5.3, constructors:
        basic_output_surface(int preferredWidth, int preferredHeight,
            io2d::format preferredFormat,
            io2d::scaling scl = io2d::scaling::letterbox,
            io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
            float fps = 30.0f);
        basic_output_surface(int preferredWidth, int preferredHeight,
            io2d::format preferredFormat,
            error_code& ec, io2d::scaling scl = io2d::scaling::letterbox,
            io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
            float fps = 30.0f) noexcept;
```

```

basic_output_surface(int preferredWidth, int preferredHeight,
    io2d::format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, io2d::format preferredDisplayFormat,
    io2d::scaling scl = io2d::scaling::letterbox,
    io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
    float fps = 30.0f);
basic_output_surface(int preferredWidth, int preferredHeight,
    io2d::format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, io2d::format preferredDisplayFormat,
    error_code& ec, io2d::scaling scl = io2d::scaling::letterbox,
    io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
    float fps = 30.0f) noexcept;

// 16.5.4, modifiers:
int begin_show();
void end_show();
void clear();
void paint(const basic_brush<GraphicsSurfaces>& b,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
template <class Allocator>
void stroke(const basic_brush<GraphicsSurfaces>& b,
    const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
    const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void stroke(const basic_brush<GraphicsSurfaces>& b,
    const basic_interpreted_path<GraphicsSurfaces>& ip,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
    const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
template <class Allocator>
void fill(const basic_brush<GraphicsSurfaces>& b,
    const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void fill(const basic_brush<GraphicsSurfaces>& b,
    const basic_interpreted_path<GraphicsSurfaces>& ip,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void mask(const basic_brush<GraphicsSurfaces>& b,
    const basic_brush<GraphicsSurfaces>& mb,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_mask_props<GraphicsSurfaces>>& mp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void draw_text(const basic_point_2d<graphics_math_type>& pt,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, const string& text,
    const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
    const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);

```

```

void draw_text(const basic_bounding_box<graphics_math_type>& bb,
  const basic_brush<GraphicsSurfaces>& b,
  const basic_font<GraphicsSurfaces>& font, const string& text,
  const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
  const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void command_list(const basic_command_list<GraphicsSurfaces>& cl);
void draw_callback(const function<void(basic_output_surface& sfc)>& fn);
void size_change_callback(
  const function<void(basic_output_surface& sfc)>& fn);
void dimensions(basic_display_point<graphics_math_type> dp);
void dimensions(basic_display_point<graphics_math_type> dp, error_code& ec) noexcept;
void output_dimensions(basic_display_point<graphics_math_type> dp);
void output_dimensions(basic_display_point<graphics_math_type> dp,
  error_code& ec) noexcept;
void scaling(io2d::scaling scl) noexcept;
void user_scaling_callback(const
  function<basic_bounding_box<graphics_math_type>(const basic_output_surface&, bool&)>& fn);
void letterbox_brush(const optional<basic_brush<GraphicsSurfaces>>& b,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt) noexcept;
void letterbox_brush_props(const optional<basic_brush_props<GraphicsSurfaces>>& bp) noexcept;
void auto_clear(bool val) noexcept;
void redraw_required(bool val = true) noexcept;

// 16.5.5, observers:
io2d::format format() const noexcept;
basic_display_point<graphics_math_type> dimensions() const noexcept;
basic_display_point<graphics_math_type> max_dimensions() const noexcept;
basic_display_point<graphics_math_type> output_dimensions() const noexcept;
basic_display_point<graphics_math_type> max_output_dimensions() const noexcept;
io2d::scaling scaling() const noexcept;
optional<basic_brush<GraphicsSurfaces>> letterbox_brush() const noexcept;
optional<basic_brush_props<GraphicsSurfaces>> letterbox_brush_props() const noexcept;
bool auto_clear() const noexcept;
};
}

```

16.5.3 basic_output_surface constructors

[io2d.outputsurface.cons]

```

basic_output_surface(int preferredWidth, int preferredHeight,
  io2d::format preferredFormat,
  io2d::scaling scl = io2d::scaling::letterbox,
  io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
  float fps = 30.0f);

```

- 1 *Effects:* Constructs an object of type `basic_output_surface`.
- 2 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_output_surface(preferredWidth, preferredHeight, preferredFormat, scl, rr, fps)`.
- 3 *Throws:* As specified in Error reporting (Clause 4).
- 4 *Error conditions:* Errors, if any, are implementation-defined

```

basic_output_surface(int preferredWidth, int preferredHeight,
  io2d::format preferredFormat,
  error_code& ec, io2d::scaling scl = io2d::scaling::letterbox,
  io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
  float fps = 30.0f) noexcept;

```

- 5 *Effects:* Constructs an object of type `basic_output_surface`.
- 6 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_output_surface(preferredWidth, preferredHeight, preferredFormat, ec, scl, rr, fps)`.

7 *Throws:* As specified in Error reporting (Clause 4).

8 *Error conditions:* Errors, if any, are implementation-defined

```
basic_output_surface(int preferredWidth, int preferredHeight,
    io2d::format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, io2d::format preferredDisplayFormat,
    io2d::scaling scl = io2d::scaling::letterbox,
    io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
    float fps = 30.0f);
```

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

10 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_output_surface(preferredWidth, preferredHeight, preferredFormat, preferredDisplayWidth, preferredDisplayHeight, preferredDisplayFormat, scl, rr, fps)`.

11 *Throws:* As specified in Error reporting (Clause 4).

12 *Error conditions:* Errors, if any, are implementation-defined

```
basic_output_surface(int preferredWidth, int preferredHeight,
    io2d::format preferredFormat, int preferredDisplayWidth,
    int preferredDisplayHeight, io2d::format preferredDisplayFormat,
    error_code& ec, io2d::scaling scl = io2d::scaling::letterbox,
    io2d::refresh_style rr = io2d::refresh_style::as_fast_as_possible,
    float fps = 30.0f) noexcept;
```

13 *Effects:* Constructs an object of type `basic_output_surface`.

14 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_output_surface(preferredWidth, preferredHeight, preferredFormat, preferredDisplayWidth, preferredDisplayHeight, preferredDisplayFormat, ec, scl, rr, fps)`.

15 *Throws:* As specified in Error reporting (Clause 4).

16 *Error conditions:* Errors, if any, are implementation-defined

16.5.4 `basic_output_surface` modifiers

[`io2d.outputsurface.modifiers`]

```
int begin_show();
```

1 *Effects:* Performs the following actions in a continuous loop:

1. Handle any implementation and host environment matters. If there are no pending implementation or host environment matters to handle, proceed immediately to the next action.
2. Run the size change callback if doing so is required by its specification and it does not have a value equivalent to its default value.
3. If the refresh style requires that the draw callback be called then:
 - a) Evaluate auto clear and perform the actions required by its specification, if any.
 - b) Run the draw callback.
 - c) Ensure that all operations from the draw callback that can effect the back buffer have completed.
 - d) Transfer the contents of the back buffer to the display buffer using sampling with an unspecified filter. If the user scaling callback does not have a value equivalent to its default value, use it to determine the position where the contents of the back buffer shall be transferred to and whether or not the letterbox brush should be used. Otherwise use the value of scaling type to determine the position and whether the letterbox brush should be used.

2 If `basic_output_surface::end_show` is called from the draw callback, the implementation shall finish executing the draw callback and shall immediately cease to perform any actions in the continuous loop other than handling any implementation and host environment matters needed to exit the loop properly.

3 No later than when this function returns, the output device shall cease to display the contents of the display buffer.

4 What the output device shall display when it is not displaying the contents of the display buffer is unspecified .

5 *Returns:* The possible values and meanings of the possible values returned are implementation-defined.

6 *Throws:* As specified in Error reporting (Clause 4).

7 *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.

8 *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.

9 Other errors, if any, produced by this function are implementation-defined.

```
void end_show();
```

10 *Effects:* If this function is called outside of the draw callback while it is being executed in the `basic_output_surface::begin_show` function's continuous loop, it does nothing.

11 Otherwise, the implementation initiates the process of exiting the `basic_output_surface::begin_show` function's continuous loop.

12 If possible, any procedures that the host environment requires in order to cause the `basic_output_surface::show` function's continuous loop to stop executing without error should be followed.

13 The `basic_output_surface::begin_show` function's loop continues execution until it returns.

```
void clear();
```

14 *Effects:* Equivalent to `paint(basic_brush<GraphicsSurfaces>(rgba_color::white), nullopt, basic_render_props<GraphicsSurfaces>(nearest, basic_matrix_2d<typename GraphicsSurfaces::graphics_math_type>{}, compositing_op::clear));`

```
void paint(const basic_brush<GraphicsSurfaces>& b,
           const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
           const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
           const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

15 *Effects:* Performs the painting rendering and compositing operation as specified by 16.3.4.

16 The meanings of the parameters are specified by 16.3.2.

17 *Throws:* As specified in Error reporting (Clause 4).

18 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
```

```
void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
            const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

```
void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
            const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

19 *Effects:* Performs the stroking rendering and compositing operation as specified by 16.3.6.

20 The meanings of the parameters are specified by 16.3.2.

21 *Throws:* As specified in Error reporting (Clause 4).

22 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```

template <class Allocator>
void fill(const basic_brush<GraphicsSurfaces>& b,
  const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void fill(const basic_brush<GraphicsSurfaces>& b,
  const basic_interpreted_path<GraphicsSurfaces>& ip,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);

```

23 *Effects:* Performs the filling rendering and composing operation as specified by [16.3.5](#).

24 The meanings of the parameters are specified by [16.3.2](#).

25 *Throws:* As specified in Error reporting ([Clause 4](#)).

26 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```

void mask(const basic_brush<GraphicsSurfaces>& b,
  const basic_brush<GraphicsSurfaces>& mb,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_mask_props<GraphicsSurfaces>>& mp = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);

```

27 *Effects:* Performs the masking rendering and composing operation as specified by [16.3.7](#).

28 The meanings of the parameters are specified by [16.3.2](#).

29 *Throws:* As specified in Error reporting ([Clause 4](#)).

30 *Error conditions:*

The errors, if any, produced by this function are implementation-defined.

```

void draw_text(const basic_point_2d<graphics_math_type>& pt,
  const basic_brush<GraphicsSurfaces>& b,
  const basic_font<GraphicsSurfaces>& font, const string& text,
  const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
  const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void draw_text(const basic_bounding_box<graphics_math_type>& bb,
  const basic_brush<GraphicsSurfaces>& b,
  const basic_font<GraphicsSurfaces>& font, const string& text,
  const optional<basic_text_props<GraphicsSurfaces>>& tp = nullopt,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
  const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);

```

31 *Effects:* Performs the text rendering rendering and composing operation as specified by [16.3.8](#).

32 The meanings of the parameters are specified by [16.3.2](#).

33 *Throws:* As specified in Error reporting ([Clause 4](#)).

34 *Error conditions:*

The errors, if any, produced by this function are implementation-defined.

```

void command_list(const basic_command_list<GraphicsSurfaces>& cl);

```

35 *Effects:* Calls `GraphicsSurfaces::surfaces::command_list(data(), cl)`.

36 *Remarks:* Stores a `basic_command_list` object for execution whenever the output needs to be redrawn.

37 If a draw callback exists, it is executed before the command list.

38 Either a draw callback or a command list must be set before `begin_show` is called. Both may be set if desired.

39 The command list may use a `run_function` command object to replace itself, but it is not permitted to replace the existing command list with a new command list more than once within the existing command list.

```
void draw_callback(const function<void(basic_output_surface& sfc)>& fn);
```

40 *Effects:* Calls `GraphicsSurfaces::surfaces::draw_callback(data(), fn)`.

41 *Remarks:* Either a draw callback or a command list must be set before `begin_show` is called. Both may be set if desired.

```
void size_change_callback(const function<void(basic_output_surface& sfc)>& fn);
```

42 *Effects:* Calls `GraphicsSurfaces::surfaces::size_change_callback(data(), fn)`.

43 *Remarks:* Sets a function that will be called whenever the output surface size changes.

```
void dimensions(basic_display_point<graphics_math_type> dp);
```

```
void dimensions(basic_display_point<graphics_math_type> dp, error_code& ec) noexcept;
```

44 *Effects:* Calls `GraphicsSurfaces::surfaces::dimensions(data(), dp)` or `GraphicsSurfaces::surfaces::dimensions(dp, ec)`.

45 *Remarks:* Changes the dimensions of the output surface's back buffer.

46 *Error conditions:* Errors, if any, are implementation-defined.

```
void display_dimensions(basic_display_point<graphics_math_type> dp);
```

```
void display_dimensions(basic_display_point<graphics_math_type> dp, error_code& ec) noexcept;
```

47 *Effects:* Calls `GraphicsSurfaces::surfaces::display_dimensions(data(), dp)` or `GraphicsSurfaces::surfaces::display_dimensions(data(), dp, ec)`.

48 *Remarks:* Changes the dimensions of the output surface's display.

49 *Error conditions:* Errors, if any, are implementation-defined.

```
void scaling(io2d::scaling scl) noexcept;
```

50 *Effects:* Calls `GraphicsSurfaces::surfaces::scaling(data(), scl)`.

51 *Remarks:* Sets the type of scaling that should be performed, if required, when transferring the graphics data from the back buffer to the display buffer.

```
void user_scaling_callback(const
```

```
function<basic_bounding_box<graphics_math_type>(const basic_output_surface&, bool&)>& fn);
```

52 *Effects:* Calls `GraphicsSurfaces::surfaces::user_scaling_callback(data(), fn)`.

53 *Remarks:* Sets an optional user function that allows the user to provide a `basic_bounding_box` object that specifies the area in the display buffer that the back buffer shall be transferred to.

```
void letterbox_brush(const optional<basic_brush<GraphicsSurfaces>>& b,
```

```
const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt) noexcept;
```

54 *Effects:* Calls `GraphicsSurfaces::surfaces::letterbox_brush(data(), b, bp)`.

```
void letterbox_brush_props(
```

```
const optional<basic_brush_props<GraphicsSurfaces>>& bp) noexcept;
```

55 *Effects:* Calls `GraphicsSurfaces::surfaces::letterbox_brush_props(data(), bp)`.

```
void auto_clear(bool val) noexcept;
```

56 *Effects:* Calls `GraphicsSurfaces::surfaces::auto_clear(data(), val)`.

```
void redraw_required(bool val = true) noexcept;
```

57 *Effects:* Calls `GraphicsSurfaces::surfaces::redraw_required(data(), val)`.

16.5.5 basic_output_surface observers [io2d.outputsurface.observers]

```

io2d::format format() const noexcept;
1     Returns: GraphicsSurfaces::surfaces::format(data()).
2     Remarks: The pixel format of the back buffer.

basic_display_point<graphics_math_type> dimensions() const noexcept;
3     Returns: GraphicsSurfaces::surfaces::dimensions(data()).
4     Remarks: The pixel dimensions of the back buffer.

basic_display_point<graphics_math_type> max_dimensions() const noexcept;
5     Returns: GraphicsSurfaces::surfaces::max_dimensions(data()).
6     Remarks: The maximum possible pixel dimensions of the back buffer.

basic_display_point<graphics_math_type> display_dimensions() const noexcept;
7     Returns: GraphicsSurfaces::surfaces::display_dimensions(data()).
8     Remarks: The pixel dimensions of the output buffer.

basic_display_point<graphics_math_type> max_output_dimensions() const noexcept;
9     Returns: GraphicsSurfaces::surfaces::max_output_dimensions(data()).
10    Remarks: The maximum possible pixel dimensions of the output buffer.

io2d::scaling scaling() const noexcept;
11    Returns: GraphicsSurfaces::surfaces::scaling(data()).
12    Remarks: The scaling type.

optional<basic_brush<GraphicsSurfaces>> letterbox_brush() const noexcept;
13    Returns: GraphicsSurfaces::surfaces::letterbox_brush(data()).
14    Remarks: An optional<basic_brush<GraphicsSurfaces>> object constructed using the user-provided
letterbox brush or, if the letterbox brush is set to its default value, an empty optional<basic_
brush<GraphicsSurfaces>> object.

optional<basic_brush_props<GraphicsSurfaces>> letterbox_brush_props() const noexcept;
15    Returns: An optional<basic_brush_props<GraphicsSurfaces>> object constructed using the user-
provided letterbox brush props or, if the letterbox brush props is set to its default value, an empty
optional<basic_brush_props<GraphicsSurfaces>> object.

bool auto_clear() const noexcept;
16    Returns: GraphicsSurfaces::surfaces::auto_clear(data()).
17    Remarks: The value of auto clear.

```

16.6 Class template basic_unmanaged_output_surface [io2d.unmanagedoutputsurface]

16.6.1 basic_unmanaged_output_surface summary [io2d.unmanagedoutputsurface.summary]

- 1 The `basic_unmanaged_output_surface` provides users the ability to use this library to draw on an existing surface, one that is not owned or managed by the library.
- 2 Specifics of its implementation depend to a certain degree on the environment, especially when it comes to constructing one. Back ends are not required to support them but should do so where possible. Once a `basic_unmanaged_output_surface` object is created, the user is able to rely on a standard API such that the only non-standard (i.e. platform and implementation-dependent) aspect is instantiation of the object.
- 3 Its data are unspecified.
- 4 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::unmanaged_output_surface_data_type`. It is accessible using the data member functions.

16.6.2 basic_unmanaged_output_surface synopsis

[io2d.unmanagedoutputsurface.synopsis]

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_unmanaged_output_surface {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surfaces::unmanaged_output_surface_data_type;

        // 16.6.3, constructor:
        basic_unmanaged_output_surface(data_type&& data) noexcept;

        // 16.6.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 16.6.5, observers:
        bool has_draw_callback() const noexcept;
        bool has_size_change_callback() const noexcept;
        bool has_user_scaling_callback() const noexcept;
        io2d::format format() const noexcept;
        basic_display_point<graphics_math_type> dimensions() const noexcept;
        basic_display_point<graphics_math_type> max_dimensions() const noexcept;
        basic_display_point<graphics_math_type> display_dimensions() const noexcept;
        basic_display_point<graphics_math_type> max_display_dimensions() const
            noexcept;
        io2d::scaling scaling() const noexcept;
        optional<basic_brush<GraphicsSurfaces>> letterbox_brush() const noexcept;
        optional<basic_brush_props<GraphicsSurfaces>> letterbox_brush_props() const
            noexcept;
        bool auto_clear() const noexcept;

        // 16.6.6, modifiers:
        void invoke_draw_callback();
        void invoke_size_change_callback();
        void draw_to_output();
        void clear();
        void paint(const basic_brush<GraphicsSurfaces>& b,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
        template <class Allocator>
        void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
            const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
        void stroke(const basic_brush<GraphicsSurfaces>& b,
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
            const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
            const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
        template <class Allocator>
        void fill(const basic_brush<GraphicsSurfaces>& b,
            const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
            const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
            const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
            const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,

```

```

    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void fill(const basic_brush<GraphicsSurfaces>& b,
    const basic_interpreted_path<GraphicsSurfaces>& ip,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void mask(const basic_brush<GraphicsSurfaces>& b,
    const basic_brush<GraphicsSurfaces>& mb,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
    const optional<basic_mask_props<GraphicsSurfaces>>& mp = nullopt,
    const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
    const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void draw_callback(
    const function<void(basic_unmanaged_output_surface& sfc)>& fn);
void size_change_callback(
    const function<void(basic_unmanaged_output_surface& sfc)>& fn);
void dimensions(basic_display_point<graphics_math_type> dp);
void dimensions(basic_display_point<graphics_math_type> dp, error_code& ec)
    noexcept;
void display_dimensions(basic_display_point<graphics_math_type> dp);
void display_dimensions(basic_display_point<graphics_math_type> dp,
    error_code& ec) noexcept;
void scaling(io2d::scaling scl) noexcept;
void letterbox_brush(const optional<basic_brush<GraphicsSurfaces>>& b,
    const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt)
    noexcept;
void letterbox_brush_props(
    const optional<basic_brush_props<GraphicsSurfaces>>& bp) noexcept;
void auto_clear(bool val) noexcept;
void redraw_required(bool val = true) noexcept;
};
}

```

16.6.3 basic_unmanaged_output_surface constructor [io2d.unmanagedoutputsurface.cons]

```
basic_unmanaged_output_surface(data_type&& data) noexcept;
```

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

2 *Remarks:* The method of constructing an object of type `data_type`, including its arguments, is implementation-defined.

3 Implementations are not required to provide this class and may explicitly do so by not providing any public constructors for `data_type`.

16.6.4 Accessors [io2d.unmanagedoutputsurface.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `basic_unmanaged_output_surface` object's data object (See: 16.6.1).

16.6.5 basic_unmanaged_output_surface observers [io2d.unmanagedoutputsurface.observers]

```
bool has_draw_callback() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::has_draw_callback(data())`

```
bool has_size_change_callback() const noexcept;
```

2 *Returns:* <TODO>

```
io2d::format format() const noexcept;
```

3 *Returns:* <TODO>

```

basic_display_point<graphics_math_type> dimensions() const noexcept;
4     Returns: <TODO>

basic_display_point<graphics_math_type> max_dimensions() const noexcept;
5     Returns: <TODO>

basic_display_point<graphics_math_type> display_dimensions() const noexcept;
6     Returns: <TODO>

basic_display_point<graphics_math_type> max_display_dimensions() const noexcept;
7     Returns: <TODO>

io2d::scaling scaling() const noexcept;
8     Returns: <TODO>

optional<basic_brush<GraphicsSurfaces>> letterbox_brush() const noexcept;
9     Returns: <TODO>

optional<basic_brush_props<GraphicsSurfaces>> letterbox_brush_props() const
noexcept;
10    Returns: <TODO>

bool auto_clear() const noexcept;
11    Returns: <TODO>

```

16.6.6 basic_unmanaged_output_surface modifiers [io2d.unmanagedoutputsurface.modifiers]

```

void clear();
1     Effects: Equivalent to paint(basic_brush<GraphicsSurfaces(rgba_color::white), nullopt,
basic_render_props<GraphicsSurfaces>(nearest, basic_matrix_2d<typename GraphicsSurfaces::graphics
math_type>{}, compositing_op::clear));

void paint(const basic_brush<GraphicsSurfaces>& b,
const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
2     Effects: Performs the painting rendering and compositing operation as specified by 16.3.4.
3     The meanings of the parameters are specified by 16.3.2.
4     Throws: As specified in Error reporting (Clause 4).
5     Error conditions: The errors, if any, produced by this function are implementation-defined.

template <class Allocator>
void stroke(const basic_brush<GraphicsSurfaces>& b,
const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void stroke(const basic_brush<GraphicsSurfaces>& b,
const basic_interpreted_path<GraphicsSurfaces>& ip,
const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
const optional<basic_stroke_props<GraphicsSurfaces>>& sp = nullopt,
const optional<basic_dashes<GraphicsSurfaces>>& d = nullopt,
const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
6     Effects: Performs the stroking rendering and compositing operation as specified by 16.3.6.

```

- 7 The meanings of the parameters are specified by [16.3.2](#).
 8 *Throws:* As specified in Error reporting ([Clause 4](#)).
 9 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
template <class Allocator>
void fill(const basic_brush<GraphicsSurfaces>& b,
  const basic_path_builder<GraphicsSurfaces, Allocator>& pb,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
void fill(const basic_brush<GraphicsSurfaces>& b,
  const basic_interpreted_path<GraphicsSurfaces>& ip,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_fill_props<GraphicsSurfaces>>& fp = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

- 10 *Effects:* Performs the filling rendering and composing operation as specified by [16.3.5](#).
 11 The meanings of the parameters are specified by [16.3.2](#).
 12 *Throws:* As specified in Error reporting ([Clause 4](#)).
 13 *Error conditions:* The errors, if any, produced by this function are implementation-defined.

```
void mask(const basic_brush<GraphicsSurfaces>& b,
  const basic_brush<GraphicsSurfaces>& mb,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt,
  const optional<basic_mask_props<GraphicsSurfaces>>& mp = nullopt,
  const optional<basic_render_props<GraphicsSurfaces>>& rp = nullopt,
  const optional<basic_clip_props<GraphicsSurfaces>>& cl = nullopt);
```

- 14 *Effects:* Performs the masking rendering and composing operation as specified by [16.3.7](#).
 15 The meanings of the parameters are specified by [16.3.2](#).
 16 *Throws:* As specified in Error reporting ([Clause 4](#)).
 17 *Error conditions:*
 The errors, if any, produced by this function are implementation-defined.

```
void draw_callback(const function<void(basic_unmanaged_output_surface& sfc)>& fn);
```

- 18 *Effects:* <TODO>

```
void size_change_callback(const function<void(basic_unmanaged_output_surface& sfc)>& fn);
```

- 19 *Effects:* <TODO>

```
void dimensions(basic_display_point<graphics_math_type> dp);
void dimensions(basic_display_point<graphics_math_type> dp, error_code& ec) noexcept;
```

- 20 *Effects:* <TODO>

```
void display_dimensions(basic_display_point<graphics_math_type> dp);
void display_dimensions(basic_display_point<graphics_math_type> dp, error_code& ec) noexcept;
```

- 21 *Effects:* <TODO>

```
void scaling(io2d::scaling scl) noexcept;
```

- 22 *Effects:* <TODO>

```
void letterbox_brush(const optional<basic_brush<GraphicsSurfaces>>& b,
  const optional<basic_brush_props<GraphicsSurfaces>>& bp = nullopt) noexcept;
void letterbox_brush_props(const optional<basic_brush_props<GraphicsSurfaces>>& bp) noexcept;
```

- 23 *Effects:* <TODO>

```
void auto_clear(bool val) noexcept;
```

²⁴ *Effects:* <TODO>

17 Command lists

[io2d.cmdlists]

17.1 Overview of command lists

[io2d.cmdlists.overview]

- 1 Command lists define operations on surfaces, *commands*, that can be submitted to a surface.
- 2 Commands consist of the rendering and compositing operations, other operations on surfaces, and a type that allows a user-provided function to run.
- 3 Command lists provide a mechanism for efficiently processing graphics operations, allowing them to be executed on multiple threads. Additionally, the `basic_interpreted_command_list` class template allows command lists to be pre-compiled by the back end, which provides optimization possibilities for back ends that use graphics acceleration hardware.

17.2 Class template `basic_commands`

[io2d.cmdlists.commands]

17.2.1 Class template `basic_commands<GraphicsSurfaces>::clear`

[io2d.cmdlists.clear]

17.2.1.1 Overview

[io2d.cmdlists.clear.intro]

- 1 The class template `basic_commands<GraphicsSurfaces>::clear` describes a command that invokes the `clear` member function of a surface.
- 2 It has an *optional surface* of type `optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>`. If optional surface has a value, the clear operation is performed on the optional surface instead of the surface that the command is submitted to.
- 3 If optional surface has a value and the referenced `basic_image_surface<GraphicsSurfaces>` object has been destroyed or otherwise rendered invalid when a `basic_command_list<GraphicsSurfaces>` object built using this `paint` object is used by the program, the effects are undefined.
- 4 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::clear_data_type`. It is accessible using the `data` member functions.

17.2.1.2 Synopsis

[io2d.cmdlists.clear.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::clear {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::surfaces::clear_data_type;

        // 17.2.1.3, construct:
        clear() noexcept;
        clear(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc)
            noexcept;

        // 17.2.1.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 17.2.1.5, modifiers:
        void surface(
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;

        // 17.2.1.6, observers:
        optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
            surface() const noexcept;
    };
};
```

```

// 17.2.1.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::clear& lhs,
    const typename basic_commands<GraphicsSurfaces::clear& rhs)
noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::clear& lhs,
    const typename basic_commands<GraphicsSurfaces::clear& rhs)
noexcept;
}

```

17.2.1.3 Constructors [io2d.cmdlists.clear.ctor]

```
clear();
```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_clear()`.

17.2.1.4 Accessors [io2d.cmdlists.clear.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `clear` object's data object (See: 17.2.1.1).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

17.2.1.5 Modifiers [io2d.cmdlists.clear.mod]

```
void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::surfaces::surface(data(), sfc)`.

2 *Remarks:* The optional surface is `sfc`.

17.2.1.6 Observers [io2d.cmdlists.clear.obs]

```
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
surface() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::surface(data())`.

2 *Remarks:* The returned value is the optional surface.

17.2.1.7 Equality operators [io2d.cmdlists.clear.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::clear& lhs,
    const typename basic_commands<GraphicsSurfaces::clear& rhs)
noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data())`.

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::clear& lhs,
    const typename basic_commands<GraphicsSurfaces::clear& rhs)
noexcept;
```

2 *Returns:* `GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data())`.

17.2.2 Class template `basic_commands<GraphicsSurfaces>::paint` `[io2d.cmdlists.commands.paint]`

17.2.2.1 Overview

`[io2d.cmdlists.paint.intro]`

- 1 The class template `basic_commands<GraphicsSurfaces>::paint` describes a command that invokes the `paint` member function of a surface.
- 2 It has an *optional surface* of type `optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>`. If optional surface has a value, the paint operation is performed on the optional surface instead of the surface that the command list is submitted to.
- 3 If optional surface has a value and the referenced `basic_image_surface<GraphicsSurfaces>` object has been destroyed or otherwise rendered invalid when a `basic_command_list<GraphicsSurfaces>` object built using this `paint` object is used by the program, the effects are undefined.
- 4 It has a *brush* of type `basic_brush<GraphicsSurfaces>`.
- 5 It has a *brush props* of type `basic_brush_props<GraphicsSurfaces>`.
- 6 It has a *render props* of type `basic_render_props<GraphicsSurfaces>`.
- 7 It has a *clip props* of type `basic_clip_props`.
- 8 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::paint_data_type`. It is accessible using the data member functions.
- 9 The data are used as arguments for the invocation of the `paint` member function of the appropriate surface when a `basic_command_list<GraphicsSurfaces>` object built using this `paint` object is used by the program.

17.2.2.2 Synopsis

`[io2d.cmdlists.paint.synopsis]`

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::paint {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::surfaces::paint_data_type;

        // 17.2.2.3, construct:
        paint(const basic_brush<GraphicsSurfaces>& b,
            const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
            const basic_render_props<GraphicsSurfaces>& rp =
                basic_render_props<GraphicsSurfaces>{},
            const basic_clip_props<GraphicsSurfaces>& cl =
                basic_clip_props<GraphicsSurfaces>{}) noexcept;
        paint(reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc,
            const basic_brush<GraphicsSurfaces>& b,
            const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
            const basic_render_props<GraphicsSurfaces>& rp =
                basic_render_props<GraphicsSurfaces>{},
            const basic_clip_props<GraphicsSurfaces>& cl =
                basic_clip_props<GraphicsSurfaces>{}) noexcept;

        // 17.2.2.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 17.2.2.5, modifiers:
        void surface(
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;
        void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
        void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
        void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
        void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;
    };
}
```

```

// 17.2.2.6, observers:
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
basic_brush<GraphicsSurfaces> brush() const noexcept;
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
};

// 17.2.2.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::paint& lhs,
    const typename basic_commands<GraphicsSurfaces::paint& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::paint& lhs,
    const typename basic_commands<GraphicsSurfaces::paint& rhs)
    noexcept;
}

```

17.2.2.3 Constructors

[io2d.cmdlists.paint.ctor]

```

paint(const basic_brush<GraphicsSurfaces>& b,
    const basic_brush_props<GraphicsSurfaces>& bp =
    basic_brush_props<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

2 *Postconditions:* data() == GraphicsSurfaces::surfaces::create_paint(b, bp, rp, cl).

```

paint(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_brush_props<GraphicsSurfaces>& bp =
    basic_brush_props<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

4 *Postconditions:* data() == GraphicsSurfaces::surfaces::create_paint(sfc, b, bp, rp, cl).

17.2.2.4 Accessors

[io2d.cmdlists.paint.acc]

```

const data_type& data() const noexcept;
data_type& data() noexcept;

```

1 *Returns:* A reference to the paint object's data object (See: 17.2.2.1).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the data_type object returned by this function.

17.2.2.5 Modifiers

[io2d.cmdlists.paint.mod]

```

void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;

```

1 *Effects:* Calls GraphicsSurfaces::surfaces::surface(data(), sfc).

2 *Remarks:* The optional surface is sfc.

```
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surfaces::brush(data(), b)`.

4 *Remarks:* The brush is `b`.

```
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
```

5 *Effects:* Calls `GraphicsSurfaces::surfaces::brush_props(data(), bp)`.

6 *Remarks:* The brush props is `bp`.

```
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
```

7 *Effects:* Calls `GraphicsSurfaces::surfaces::render_props(data(), rp)`.

8 *Remarks:* The render props is `rp`.

```
void clip_props(const basic_clip_props<GraphicsSurfaces>& c1) noexcept;
```

9 *Effects:* Calls `GraphicsSurfaces::surfaces::clip_props(data(), c1)`.

10 *Remarks:* The clip props is `c1`.

17.2.2.6 Observers

[io2d.cmdlists.paint.obs]

```
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
  surface() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::surface(data())`.

2 *Remarks:* The returned value is the optional surface.

```
basic_brush<GraphicsSurfaces> brush() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surfaces::brush(data())`.

4 *Remarks:* The returned value is the brush.

```
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
```

5 *Returns:* `GraphicsSurfaces::surfaces::brush_props(data())`.

6 *Remarks:* The returned value is the brush props.

```
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
```

7 *Returns:* `GraphicsSurfaces::surfaces::render_props(data())`.

8 *Remarks:* The returned value is the render props.

```
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
```

9 *Returns:* `GraphicsSurfaces::surfaces::clip_props(data())`.

10 *Remarks:* The returned value is the clip props.

17.2.2.7 Equality operators

[io2d.cmdlists.paint.eq]

```
template <class GraphicsSurfaces>
bool operator==(
  const typename basic_commands<GraphicsSurfaces::paint& lhs,
  const typename basic_commands<GraphicsSurfaces::paint& rhs)
  noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data())`.

```
template <class GraphicsSurfaces>
bool operator!=(
  const typename basic_commands<GraphicsSurfaces::paint& lhs,
  const typename basic_commands<GraphicsSurfaces::paint& rhs)
  noexcept;
```

2 *Returns:* `GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data())`.

17.2.3 Class template `basic_commands<GraphicsSurfaces>::stroke` [io2d.cmdlists.commands.stroke]

17.2.3.1 Overview [io2d.cmdlists.stroke.intro]

- ¹ The class template `basic_commands<GraphicsSurfaces>::stroke` describes a command that invokes the `stroke` member function of a surface.
- ² It has an *optional surface* of type `optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>`. If optional surface has a value, the stroke operation is performed on the optional surface instead of the surface that the command is submitted to.
- ³ If optional surface has a value and the referenced `basic_image_surface<GraphicsSurfaces>` object has been destroyed or otherwise rendered invalid when a `basic_command_list<GraphicsSurfaces>` object built using this paint object is used by the program, the effects are undefined.
- ⁴ It has a *brush* of type `basic_brush<GraphicsSurfaces>`.
- ⁵ It has a *path* of type `basic_interpreted_path<GraphicsSurfaces>`.
- ⁶ It has a *brush props* of type `basic_brush_props<GraphicsSurfaces>`.
- ⁷ It has a *stroke props* of type `basic_stroke_props<GraphicsSurfaces>`.
- ⁸ It has a *dashes* of type `basic_dashes<GraphicsSurfaces>`.
- ⁹ It has a *render props* of type `basic_render_props<GraphicsSurfaces>`.
- ¹⁰ It has a *clip props* of type `basic_clip_props`.
- ¹¹ The data are stored in an object of type `typename GraphicsSurfaces::surfaces::stroke_data_type`. It is accessible using the data member functions.
- ¹² The data are used as arguments for the invocation of the `stroke` member function of the appropriate surface when a `basic_command_list<GraphicsSurfaces>` object built using this stroke object is used by the program.

17.2.3.2 Synopsis [io2d.cmdlists.stroke.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::stroke {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::surfaces::stroke_data_type;

        // 17.2.3.3, construct:
        stroke(const basic_brush<GraphicsSurfaces>& b,
              const basic_interpreted_path<GraphicsSurfaces>& ip,
              const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
              const basic_stroke_props<GraphicsSurfaces>& sp =
                basic_stroke_props<GraphicsSurfaces>{},
              const basic_dashes<GraphicsSurfaces>& d =
                basic_dashes<GraphicsSurfaces>{},
              const basic_render_props<GraphicsSurfaces>& rp =
                basic_render_props<GraphicsSurfaces>{},
              const basic_clip_props<GraphicsSurfaces>& cl =
                basic_clip_props<GraphicsSurfaces>{}) noexcept;
        stroke(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
              const basic_brush<GraphicsSurfaces>& b,
              const basic_interpreted_path<GraphicsSurfaces>& ip,
              const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
              const basic_stroke_props<GraphicsSurfaces>& sp =
                basic_stroke_props<GraphicsSurfaces>{},
              const basic_dashes<GraphicsSurfaces>& d =
                basic_dashes<GraphicsSurfaces>{},
              const basic_render_props<GraphicsSurfaces>& rp =
                basic_render_props<GraphicsSurfaces>{}),
```

```

    const basic_clip_props<GraphicsSurfaces>& cl =
        basic_clip_props<GraphicsSurfaces>{ }) noexcept;

// 17.2.3.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 17.2.3.5, modifiers:
void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
void path(const basic_interpreted_path<GraphicsSurfaces>& p) noexcept;
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
void stroke_props(const basic_stroke_props<GraphicsSurfaces>& sp) noexcept;
void dashes(const basic_dashes<GraphicsSurfaces>& d) noexcept;
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;

// 17.2.3.6, observers:
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
basic_brush<GraphicsSurfaces> brush() const noexcept;
basic_interpreted_path<GraphicsSurfaces> path() const noexcept;
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
basic_stroke_props<GraphicsSurfaces> stroke_props() const noexcept;
basic_dashes<GraphicsSurfaces> dashes() const noexcept;
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
};

// 17.2.3.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::stroke& lhs,
    const typename basic_commands<GraphicsSurfaces::stroke& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::stroke& lhs,
    const typename basic_commands<GraphicsSurfaces::stroke& rhs)
    noexcept;
}

```

17.2.3.3 Constructors

[io2d.cmdlists.stroke.ctor]

```

stroke(const basic_brush<GraphicsSurfaces>& b,
    const basic_interpreted_path<GraphicsSurfaces>& ip,
    const basic_brush_props<GraphicsSurfaces>& bp =
        basic_brush_props<GraphicsSurfaces>{ },
    const basic_stroke_props<GraphicsSurfaces>& sp =
        basic_stroke_props<GraphicsSurfaces>{ },
    const basic_dashes<GraphicsSurfaces>& d =
        basic_dashes<GraphicsSurfaces>{ },
    const basic_render_props<GraphicsSurfaces>& rp =
        basic_render_props<GraphicsSurfaces>{ },
    const basic_clip_props<GraphicsSurfaces>& cl =
        basic_clip_props<GraphicsSurfaces>{ }) noexcept;

```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_stroke(b, ip, bp, sp, d, rp, cl)`.

```
stroke(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
  const basic_brush<GraphicsSurfaces>& b,
  const basic_interpreted_path<GraphicsSurfaces>& ip,
  const basic_brush_props<GraphicsSurfaces>& bp =
  basic_brush_props<GraphicsSurfaces>{},
  const basic_stroke_props<GraphicsSurfaces>& sp =
  basic_stroke_props<GraphicsSurfaces>{},
  const basic_dashes<GraphicsSurfaces>& d =
  basic_dashes<GraphicsSurfaces>{},
  const basic_render_props<GraphicsSurfaces>& rp =
  basic_render_props<GraphicsSurfaces>{},
  const basic_clip_props<GraphicsSurfaces>& cl =
  basic_clip_props<GraphicsSurfaces>{}) noexcept;
```

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

4 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_stroke(sfc, b, ip, bp, sp, d, rp, cl)`.

17.2.3.4 Accessors

[io2d.cmdlists.stroke.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `stroke` object's data object (See: [17.2.3.1](#)).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

17.2.3.5 Modifiers

[io2d.cmdlists.stroke.mod]

```
void surface(
  optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
  noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::surfaces::surface(data(), sfc)`.

2 *Remarks:* The optional surface is `sfc`.

```
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surfaces::brush(data(), b)`.

4 *Remarks:* The brush is `b`.

```
void path(const basic_interpreted_path<GraphicsSurfaces>& p) noexcept;
```

5 *Effects:* Calls `GraphicsSurfaces::surfaces::path(data(), p)`.

6 *Remarks:* The path is `p`.

```
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
```

7 *Effects:* Calls `GraphicsSurfaces::surfaces::brush_props(data(), bp)`.

8 *Remarks:* The brush props is `bp`.

```
void brush_props(const basic_stroke_props<GraphicsSurfaces>& sp) noexcept;
```

9 *Effects:* Calls `GraphicsSurfaces::surfaces::stroke_props(data(), sp)`.

10 *Remarks:* The stroke props is `sp`.

```
void dashes(const basic_dashes<GraphicsSurfaces>& d) noexcept;
```

11 *Effects:* Calls `GraphicsSurfaces::surfaces::dashes(data(), d)`.

12 *Remarks:* The dashes is `d`.

```
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
```

13 *Effects:* Calls `GraphicsSurfaces::surfaces::render_props(data(), rp)`.

14 *Remarks:* The render props is `rp`.

```
void clip_props(const basic_clip_props<GraphicsSurfaces>& c1) noexcept;
```

15 *Effects:* Calls `GraphicsSurfaces::surfaces::clip_props(data(), c1)`.

16 *Remarks:* The clip props is `c1`.

17.2.3.6 Observers

[io2d.cmdlists.stroke.obs]

```
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
```

```
surface() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::surface(data())`.

2 *Remarks:* The returned value is the optional surface.

```
basic_brush<GraphicsSurfaces> brush() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surfaces::brush(data())`.

4 *Remarks:* The returned value is the brush.

```
basic_interpreted_path<GraphicsSurfaces> path() const noexcept;
```

5 *Returns:* `GraphicsSurfaces::surfaces::path(data())`.

6 *Remarks:* The returned value is the path.

```
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
```

7 *Returns:* `GraphicsSurfaces::surfaces::brush_props(data())`.

8 *Remarks:* The returned value is the brush props.

```
basic_stroke_props<GraphicsSurfaces> stroke_props() const noexcept;
```

9 *Returns:* `GraphicsSurfaces::surfaces::stroke_props(data())`.

10 *Remarks:* The returned value is the stroke props.

```
basic_dashes<GraphicsSurfaces> dashes() const noexcept;
```

11 *Returns:* `GraphicsSurfaces::surfaces::dashes(data())`.

12 *Remarks:* The returned value is the dashes.

```
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
```

13 *Returns:* `GraphicsSurfaces::surfaces::render_props(data())`.

14 *Remarks:* The returned value is the render props.

```
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
```

15 *Returns:* `GraphicsSurfaces::surfaces::clip_props(data())`.

16 *Remarks:* The returned value is the clip props.

17.2.3.7 Equality operators

[io2d.cmdlists.stroke.eq]

```
template <class GraphicsSurfaces>
```

```
bool operator==(
```

```
const typename basic_commands<GraphicsSurfaces::stroke& lhs,
```

```
const typename basic_commands<GraphicsSurfaces::stroke& rhs)
```

```
noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data())`.

```
template <class GraphicsSurfaces>
```

```
bool operator!=(
```

```
const typename basic_commands<GraphicsSurfaces::stroke& lhs,
```

```
const typename basic_commands<GraphicsSurfaces::stroke& rhs)
```

```
noexcept;
```

2 *Returns:* `GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data())`.

17.2.4 Class template `basic_commands<GraphicsSurfaces>::fill` [io2d.cmdlists.commands.fill]

17.2.4.1 Overview

[io2d.cmdlists.fill.intro]

- 1 The class template `basic_commands<GraphicsSurfaces>::fill` describes a command that invokes the `fill` member function of a surface.
- 2 It has an *optional surface* of type `optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>`. If optional surface has a value, the fill operation is performed on the optional surface instead of the surface that the command is submitted to.
- 3 If optional surface has a value and the referenced `basic_image_surface<GraphicsSurfaces>` object has been destroyed or otherwise rendered invalid when a `basic_command_list<GraphicsSurfaces>` object built using this `paint` object is used by the program, the effects are undefined.
- 4 It has a *brush* of type `basic_brush<GraphicsSurfaces>`.
- 5 It has a *path* of type `basic_interpreted_path<GraphicsSurfaces>`.
- 6 It has a *brush props* of type `basic_brush_props<GraphicsSurfaces>`.
- 7 It has a *render props* of type `basic_render_props<GraphicsSurfaces>`.
- 8 It has a *clip props* of type `basic_clip_props`.
- 9 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::fill_data_type`. It is accessible using the `data` member functions.
- 10 The data are used as arguments for the invocation of the `fill` member function of the appropriate surface when a `basic_command_list<GraphicsSurfaces>` object built using this `fill` object is used by the program.

17.2.4.2 Synopsis

[io2d.cmdlists.fill.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::fill {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::surfaces::fill_data_type;

        // 17.2.4.3, construct:
        fill(const basic_brush<GraphicsSurfaces>& b,
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
            const basic_render_props<GraphicsSurfaces>& rp =
                basic_render_props<GraphicsSurfaces>{},
            const basic_clip_props<GraphicsSurfaces>& cl =
                basic_clip_props<GraphicsSurfaces>{}) noexcept;
        fill(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
            const basic_brush<GraphicsSurfaces>& b,
            const basic_interpreted_path<GraphicsSurfaces>& ip,
            const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
            const basic_render_props<GraphicsSurfaces>& rp =
                basic_render_props<GraphicsSurfaces>{},
            const basic_clip_props<GraphicsSurfaces>& cl =
                basic_clip_props<GraphicsSurfaces>{}) noexcept;

        // 17.2.4.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 17.2.4.5, modifiers:
        void surface(
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;
        void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
        void path(const basic_interpreted_path<GraphicsSurfaces>& p) noexcept;
```



```

void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
void clip_props(const basic_clip_props<GraphicsSurfaces>& c1) noexcept;

// 17.2.4.6, observers:
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
basic_brush<GraphicsSurfaces> brush() const noexcept;
basic_interpreted_path<GraphicsSurfaces> path() const noexcept;
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
};

// 17.2.4.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::fill>& lhs,
    const typename basic_commands<GraphicsSurfaces::fill>& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::fill>& lhs,
    const typename basic_commands<GraphicsSurfaces::fill>& rhs)
    noexcept;
}

```

17.2.4.3 Constructors

[io2d.cmdlists.fill.ctor]

```

fill(const basic_brush<GraphicsSurfaces>& b,
     const basic_interpreted_path<GraphicsSurfaces>& ip,
     const basic_brush_props<GraphicsSurfaces>& bp =
         basic_brush_props<GraphicsSurfaces>{},
     const basic_render_props<GraphicsSurfaces>& rp =
         basic_render_props<GraphicsSurfaces>{},
     const basic_clip_props<GraphicsSurfaces>& c1 =
         basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

2 *Postconditions:* data() == GraphicsSurfaces::surfaces::create_fill(b, ip, bp, rp, c1).

```

fill(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
     const basic_brush<GraphicsSurfaces>& b,
     const basic_interpreted_path<GraphicsSurfaces>& ip,
     const basic_brush_props<GraphicsSurfaces>& bp =
         basic_brush_props<GraphicsSurfaces>{},
     const basic_render_props<GraphicsSurfaces>& rp =
         basic_render_props<GraphicsSurfaces>{},
     const basic_clip_props<GraphicsSurfaces>& c1 =
         basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

4 *Postconditions:* data() == GraphicsSurfaces::surfaces::create_fill(sfc, b, ip, bp, rp, c1).

17.2.4.4 Accessors

[io2d.cmdlists.fill.acc]

```

const data_type& data() const noexcept;
data_type& data() noexcept;

```

1 *Returns:* A reference to the fill object's data object (See: [17.2.4.1](#)).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the data_type object returned by this function.

17.2.4.5 Modifiers

[io2d.cmdlists.fill.mod]

```
void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::surfaces::surface(data(), sfc)`.

2 *Remarks:* The optional surface is `sfc`.

```
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surfaces::brush(data(), b)`.

4 *Remarks:* The brush is `b`.

```
void path(const basic_interpreted_path<GraphicsSurfaces>& p) noexcept;
```

5 *Effects:* Calls `GraphicsSurfaces::surfaces::path(data(), p)`.

6 *Remarks:* The path is `p`.

```
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
```

7 *Effects:* Calls `GraphicsSurfaces::surfaces::brush_props(data(), bp)`.

8 *Remarks:* The brush props is `bp`.

```
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
```

9 *Effects:* Calls `GraphicsSurfaces::surfaces::render_props(data(), rp)`.

10 *Remarks:* The render props is `rp`.

```
void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;
```

11 *Effects:* Calls `GraphicsSurfaces::surfaces::clip_props(data(), cl)`.

12 *Remarks:* The clip props is `cl`.

17.2.4.6 Observers

[io2d.cmdlists.fill.obs]

```
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::surface(data())`.

2 *Remarks:* The returned value is the optional surface.

```
basic_brush<GraphicsSurfaces> brush() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surfaces::brush(data())`.

4 *Remarks:* The returned value is the brush.

```
basic_interpreted_path<GraphicsSurfaces> path() const noexcept;
```

5 *Returns:* `GraphicsSurfaces::surfaces::path(data())`.

6 *Remarks:* The returned value is the path.

```
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
```

7 *Returns:* `GraphicsSurfaces::surfaces::brush_props(data())`.

8 *Remarks:* The returned value is the brush props.

```
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
```

9 *Returns:* `GraphicsSurfaces::surfaces::render_props(data())`.

10 *Remarks:* The returned value is the render props.

```
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
```

11 *Returns:* `GraphicsSurfaces::surfaces::clip_props(data())`.

12 *Remarks:* The returned value is the clip props.

17.2.4.7 Equality operators

[io2d.cmdlists.fill.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::fill& lhs,
    const typename basic_commands<GraphicsSurfaces::fill& rhs)
noexcept;
```

1 *Returns:* GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data()).

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::fill& lhs,
    const typename basic_commands<GraphicsSurfaces::fill& rhs)
noexcept;
```

2 *Returns:* GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data()).

17.2.5 Class template basic_commands<GraphicsSurfaces>::mask

[io2d.cmdlists.commands.mask]

17.2.5.1 Overview

[io2d.cmdlists.mask.intro]

- 1 The class template basic_commands<GraphicsSurfaces>::mask describes a command that invokes the mask member function of a surface.
- 2 It has an *optional surface* of type optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>. If optional surface has a value, the mask operation is performed on the optional surface instead of the surface that the command is submitted to.
- 3 If optional surface has a value and the referenced basic_image_surface<GraphicsSurfaces> object has been destroyed or otherwise rendered invalid when a basic_command_list<GraphicsSurfaces> object built using this paint object is used by the program, the effects are undefined.
- 4 It has a *brush* of type basic_brush<GraphicsSurfaces>.
- 5 It has a *mask brush* of type basic_brush<GraphicsSurfaces>.
- 6 It has a *brush props* of type basic_brush_props<GraphicsSurfaces>.
- 7 It has a *mask props* of type basic_mask_props<GraphicsSurfaces>.
- 8 It has a *render props* of type basic_render_props<GraphicsSurfaces>.
- 9 It has a *clip props* of type basic_clip_props.
- 10 The data are stored in an object of type typename GraphicsSurfaces::surfaces::mask_data_type. It is accessible using the data member functions.
- 11 The data are used as arguments for the invocation of the mask member function of the appropriate surface when a basic_command_list<GraphicsSurfaces> object built using this mask object is used by the program.

17.2.5.2 Synopsis

[io2d.cmdlists.mask.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::mask {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::surfaces::mask_data_type;

        // 17.2.5.3, construct:
        mask(const basic_brush<GraphicsSurfaces>& b,
            const basic_brush<GraphicsSurfaces>& mb,
            const basic_brush_props<GraphicsSurfaces>& bp =
                basic_brush_props<GraphicsSurfaces>{},
            const basic_mask_props<GraphicsSurfaces>& mp =
                basic_mask_props<GraphicsSurfaces>{},
            const basic_render_props<GraphicsSurfaces>& rp =
                basic_mask_props<GraphicsSurfaces>{},
            const basic_clip_props<GraphicsSurfaces>& cl =
                basic_clip_props<GraphicsSurfaces>{}) noexcept;
```

```

mask(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_brush<GraphicsSurfaces>& mb,
    const basic_brush_props<GraphicsSurfaces>& bp =
        basic_brush_props<GraphicsSurfaces>{},
    const basic_mask_props<GraphicsSurfaces>& mp =
        basic_mask_props<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
        basic_mask_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
        basic_clip_props<GraphicsSurfaces>{}) noexcept;

// 17.2.5.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 17.2.5.5, modifiers:
void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
void mask_brush(const basic_brush<GraphicsSurfaces>& mb) noexcept;
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
void mask_props(const basic_mask_props<GraphicsSurfaces>& mp) noexcept;
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;

// 17.2.5.6, observers:
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
basic_brush<GraphicsSurfaces> brush() const noexcept;
basic_brush<GraphicsSurfaces> mask_brush() const noexcept;
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
basic_mask_props<GraphicsSurfaces> mask_props() const noexcept;
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
};

// 17.2.5.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::mask& lhs,
    const typename basic_commands<GraphicsSurfaces::mask& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::mask& lhs,
    const typename basic_commands<GraphicsSurfaces::mask& rhs)
    noexcept;
}

```

17.2.5.3 Constructors

[io2d.cmdlists.mask.ctor]

```

mask(const basic_brush<GraphicsSurfaces>& b,
    const basic_brush<GraphicsSurfaces>& mb,
    const basic_brush_props<GraphicsSurfaces>& bp =
        basic_brush_props<GraphicsSurfaces>{},
    const basic_mask_props<GraphicsSurfaces>& mp =
        basic_mask_props<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
        basic_mask_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
        basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_mask(b, mb, bp, mp, rp, cl)`.

```
mask(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
      const basic_brush<GraphicsSurfaces>& b,
      const basic_brush<GraphicsSurfaces>& mb,
      const basic_brush_props<GraphicsSurfaces>& bp =
        basic_brush_props<GraphicsSurfaces>{},
      const basic_mask_props<GraphicsSurfaces>& mp =
        basic_mask_props<GraphicsSurfaces>{},
      const basic_render_props<GraphicsSurfaces>& rp =
        basic_mask_props<GraphicsSurfaces>{},
      const basic_clip_props<GraphicsSurfaces>& cl =
        basic_clip_props<GraphicsSurfaces>{}) noexcept;
```

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

4 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_mask(sfc, b, mb, bp, mp, rp, cl)`.

17.2.5.4 Accessors

[io2d.cmdlists.mask.acc]

```
const data_type& data() const noexcept;
data_type& data() noexcept;
```

1 *Returns:* A reference to the `mask` object's data object (See: [17.2.5.1](#)).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

17.2.5.5 Modifiers

[io2d.cmdlists.mask.mod]

```
void surface(
  optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
  noexcept;
```

1 *Effects:* Calls `GraphicsSurfaces::surfaces::surface(data(), sfc)`.

2 *Remarks:* The optional surface is `sfc`.

```
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surfaces::brush(data(), b)`.

4 *Remarks:* The brush is `b`.

```
void path(const basic_brush<GraphicsSurfaces>& mb) noexcept;
```

5 *Effects:* Calls `GraphicsSurfaces::surfaces::mask_brush(data(), mb)`.

6 *Remarks:* The mask brush is `mb`.

```
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
```

7 *Effects:* Calls `GraphicsSurfaces::surfaces::brush_props(data(), bp)`.

8 *Remarks:* The brush props is `bp`.

```
void mask_props(const basic_mask_props<GraphicsSurfaces>& bp) noexcept;
```

9 *Effects:* Calls `GraphicsSurfaces::surfaces::mask_props(data(), bp)`.

10 *Remarks:* The mask props is `bp`.

```
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
```

11 *Effects:* Calls `GraphicsSurfaces::surfaces::render_props(data(), rp)`.

12 *Remarks:* The render props is `rp`.

```
void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;
```

13 *Effects:* Calls `GraphicsSurfaces::surfaces::clip_props(data(), cl)`.

14 *Remarks:* The clip props is `cl`.

17.2.5.6 Observers

[io2d.cmdlists.mask.obs]

```
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
  surface() const noexcept;
```

1 *Returns:* GraphicsSurfaces::surfaces::surface(data()).

2 *Remarks:* The returned value is the optional surface.

```
basic_brush<GraphicsSurfaces> brush() const noexcept;
```

3 *Returns:* GraphicsSurfaces::surfaces::brush(data()).

4 *Remarks:* The returned value is the brush.

```
basic_brush<GraphicsSurfaces> mask_brush() const noexcept;
```

5 *Returns:* GraphicsSurfaces::surfaces::mask_brush(data()).

6 *Remarks:* The returned value is the mask brush.

```
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
```

7 *Returns:* GraphicsSurfaces::surfaces::brush_props(data()).

8 *Remarks:* The returned value is the brush props.

```
basic_mask_props<GraphicsSurfaces> mask_props() const noexcept;
```

9 *Returns:* GraphicsSurfaces::surfaces::mask_props(data()).

10 *Remarks:* The returned value is the mask props.

```
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
```

11 *Returns:* GraphicsSurfaces::surfaces::render_props(data()).

12 *Remarks:* The returned value is the render props.

```
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
```

13 *Returns:* GraphicsSurfaces::surfaces::clip_props(data()).

14 *Remarks:* The returned value is the clip props.

17.2.5.7 Equality operators

[io2d.cmdlists.mask.eq]

```
template <class GraphicsSurfaces>
bool operator==(
  const typename basic_commands<GraphicsSurfaces::mask& lhs,
  const typename basic_commands<GraphicsSurfaces::mask& rhs)
  noexcept;
```

1 *Returns:* GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data()).

```
template <class GraphicsSurfaces>
bool operator!=(
  const typename basic_commands<GraphicsSurfaces::mask& lhs,
  const typename basic_commands<GraphicsSurfaces::mask& rhs)
  noexcept;
```

2 *Returns:* GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data()).

17.2.6 Class template basic_commands<GraphicsSurfaces>::draw_text

[io2d.cmdlists.commands.drawtext]

17.2.6.1 Overview

[io2d.cmdlists.drawtext.intro]

- 1 The class template `basic_commands<GraphicsSurfaces>::draw_text` describes a command that invokes the `draw_text` member function of a surface.
- 2 It has an *optional surface* of type `optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>`. If optional surface has a value, the `draw_text` operation is performed on the optional surface instead of the surface that the command is submitted to.

- 3 If optional surface has a value and the referenced `basic_image_surface<GraphicsSurfaces>` object has been destroyed or otherwise rendered invalid when a `basic_command_list<GraphicsSurfaces>` object built using this `paint` object is used by the program, the effects are undefined.
- 4 It has a *text location* of type `variant<basic_point_2d<typename GraphicsSurfaces::graphics_math_type>, basic_bounding_box<typename GraphicsSurfaces::graphics_math_type>>`.
- 5 It has a *brush* of type `basic_brush<GraphicsSurfaces>`.
- 6 It has a *font* of type `basic_font<GraphicsSurfaces>`.
- 7 It has *text* of type `string` comprised of UTF-8 encoded character data.
- 8 It has a *text props* of type `basic_text_props<GraphicsSurfaces>`.
- 9 It has a *brush props* of type `basic_brush_props<GraphicsSurfaces>`.
- 10 It has a *stroke props* of type `basic_stroke_props<GraphicsSurfaces>`.
- 11 It has a *dashes* of type `basic_dashes<GraphicsSurfaces>`.
- 12 It has a *render props* of type `basic_render_props<GraphicsSurfaces>`.
- 13 It has a *clip props* of type `basic_clip_props`.
- 14 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::stroke_data_type`. It is accessible using the data member functions.
- 15 The data are used as arguments for the invocation of the `draw_text` member function of the appropriate surface when a `basic_command_list<GraphicsSurfaces>` object built using this `draw_text` object is used by the program.

17.2.6.2 Synopsis

[io2d.cmdlists.drawtext.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::draw_text {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type = typename GraphicsSurfaces::surfaces::draw_text_data_type;

        // 17.2.6.3, construct:
        draw_text(const basic_point_2d<graphics_math_type>& pt,
                 const basic_brush<GraphicsSurfaces>& b,
                 const basic_font<GraphicsSurfaces>& font, string t,
                 const basic_text_props<GraphicsSurfaces>& tp =
                 basic_text_props<GraphicsSurfaces>{},
                 const basic_brush_props<GraphicsSurfaces>& bp =
                 basic_brush_props<GraphicsSurfaces>{},
                 const basic_stroke_props<GraphicsSurfaces>& sp =
                 basic_stroke_props<GraphicsSurfaces>{},
                 const basic_dashes<GraphicsSurfaces>& d =
                 basic_dashes<GraphicsSurfaces>{},
                 const basic_render_props<GraphicsSurfaces>& rp =
                 basic_render_props<GraphicsSurfaces>{},
                 const basic_clip_props<GraphicsSurfaces>& cl =
                 basic_clip_props<GraphicsSurfaces>{}) noexcept;
        draw_text(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
                 const basic_point_2d<graphics_math_type>& pt,
                 const basic_brush<GraphicsSurfaces>& b,
                 const basic_font<GraphicsSurfaces>& font, string t,
                 const basic_text_props<GraphicsSurfaces>& tp =
                 basic_text_props<GraphicsSurfaces>{},
                 const basic_brush_props<GraphicsSurfaces>& bp =
                 basic_brush_props<GraphicsSurfaces>{},
                 const basic_stroke_props<GraphicsSurfaces>& sp =
                 basic_stroke_props<GraphicsSurfaces>{},
                 const basic_dashes<GraphicsSurfaces>& d =
                 basic_dashes<GraphicsSurfaces>{}),
```

```

    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{} noexcept;
draw_text(const basic_bounding_box<graphics_math_type>& bb,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, string t,
    const basic_text_props<GraphicsSurfaces>& tp =
    basic_text_props<GraphicsSurfaces>{},
    const basic_brush_props<GraphicsSurfaces>& bp =
    basic_brush_props<GraphicsSurfaces>{},
    const basic_stroke_props<GraphicsSurfaces>& sp =
    basic_stroke_props<GraphicsSurfaces>{},
    const basic_dashes<GraphicsSurfaces>& d =
    basic_dashes<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{} noexcept;
draw_text(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
    const basic_bounding_box<graphics_math_type>& bb,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, string t,
    const basic_text_props<GraphicsSurfaces>& tp =
    basic_text_props<GraphicsSurfaces>{},
    const basic_brush_props<GraphicsSurfaces>& bp =
    basic_brush_props<GraphicsSurfaces>{},
    const basic_stroke_props<GraphicsSurfaces>& sp =
    basic_stroke_props<GraphicsSurfaces>{},
    const basic_dashes<GraphicsSurfaces>& d =
    basic_dashes<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{} noexcept;

// 17.2.6.4, accessors:
const data_type& data() const noexcept;
data_type& data() noexcept;

// 17.2.6.5, modifiers:
void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;
void location(const basic_point_2d<graphics_math_type>& pt) noexcept;
void location(const basic_bounding_box<graphics_math_type>& bb) noexcept;
void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
void font(const basic_font<GraphicsSurfaces>& f) noexcept;
void text(string t) noexcept;
void text_props(const basic_text_props<GraphicsSurfaces>& tp) noexcept;
void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
void stroke_props(const basic_stroke_props<GraphicsSurfaces>& sp) noexcept;
void dashes(const basic_dashes<GraphicsSurfaces>& d) noexcept;
void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;

// 17.2.6.6, observers:
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
variant<basic_point_2d<graphics_math_type>,
    basic_bounding_box<graphics_math_type>> location() const noexcept;
basic_brush<GraphicsSurfaces> brush() const noexcept;
basic_font<GraphicsSurfaces> font() const noexcept;
string text() const noexcept;

```



```

    basic_text_props<GraphicsSurfaces> text_props() const noexcept;
    basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
    basic_stroke_props<GraphicsSurfaces> stroke_props() const noexcept;
    basic_dashes<GraphicsSurfaces> dashes() const noexcept;
    basic_render_props<GraphicsSurfaces> render_props() const noexcept;
    basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
};

```

// 17.2.6.7, equality operators:

```

template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::draw_text& lhs,
    const typename basic_commands<GraphicsSurfaces::draw_text& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::draw_text& lhs,
    const typename basic_commands<GraphicsSurfaces::draw_text& rhs)
    noexcept;
}

```

17.2.6.3 Constructors

[io2d.cmdlists.drawtext.ctor]

```

draw_text(const basic_point_2d<graphics_math_type>& pt,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, string t,
    const basic_text_props<GraphicsSurfaces>& tp =
    basic_text_props<GraphicsSurfaces>{},
    const basic_brush_props<GraphicsSurfaces>& bp =
    basic_brush_props<GraphicsSurfaces>{},
    const basic_stroke_props<GraphicsSurfaces>& sp =
    basic_stroke_props<GraphicsSurfaces>{},
    const basic_dashes<GraphicsSurfaces>& d =
    basic_dashes<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_draw_text(pt, b, font, t, tp, bp, sp, d, rp, cl)`.

```

draw_text(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
    const basic_point_2d<graphics_math_type>& pt,
    const basic_brush<GraphicsSurfaces>& b,
    const basic_font<GraphicsSurfaces>& font, string t,
    const basic_text_props<GraphicsSurfaces>& tp =
    basic_text_props<GraphicsSurfaces>{},
    const basic_brush_props<GraphicsSurfaces>& bp =
    basic_brush_props<GraphicsSurfaces>{},
    const basic_stroke_props<GraphicsSurfaces>& sp =
    basic_stroke_props<GraphicsSurfaces>{},
    const basic_dashes<GraphicsSurfaces>& d =
    basic_dashes<GraphicsSurfaces>{},
    const basic_render_props<GraphicsSurfaces>& rp =
    basic_render_props<GraphicsSurfaces>{},
    const basic_clip_props<GraphicsSurfaces>& cl =
    basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

4 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_draw_text(sfc, pt, b, font, t, tp, bp, sp, d, rp, cl)`.

```

draw_text(const basic_bounding_box<graphics_math_type>& bb,
  const basic_brush<GraphicsSurfaces>& b,
  const basic_font<GraphicsSurfaces>& font, string t,
  const basic_text_props<GraphicsSurfaces>& tp =
  basic_text_props<GraphicsSurfaces>{},
  const basic_brush_props<GraphicsSurfaces>& bp =
  basic_brush_props<GraphicsSurfaces>{},
  const basic_stroke_props<GraphicsSurfaces>& sp =
  basic_stroke_props<GraphicsSurfaces>{},
  const basic_dashes<GraphicsSurfaces>& d =
  basic_dashes<GraphicsSurfaces>{},
  const basic_render_props<GraphicsSurfaces>& rp =
  basic_render_props<GraphicsSurfaces>{},
  const basic_clip_props<GraphicsSurfaces>& cl =
  basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

6 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_draw_text(bb, b, font, t, tp, bp, sp, d, rp, cl)`.

```

draw_text(reference_wrapper<basic_image_surface<GraphicsSurfaces>> sfc,
  const basic_bounding_box<graphics_math_type>& bb,
  const basic_brush<GraphicsSurfaces>& b,
  const basic_font<GraphicsSurfaces>& font, string t,
  const basic_text_props<GraphicsSurfaces>& tp =
  basic_text_props<GraphicsSurfaces>{},
  const basic_brush_props<GraphicsSurfaces>& bp =
  basic_brush_props<GraphicsSurfaces>{},
  const basic_stroke_props<GraphicsSurfaces>& sp =
  basic_stroke_props<GraphicsSurfaces>{},
  const basic_dashes<GraphicsSurfaces>& d =
  basic_dashes<GraphicsSurfaces>{},
  const basic_render_props<GraphicsSurfaces>& rp =
  basic_render_props<GraphicsSurfaces>{},
  const basic_clip_props<GraphicsSurfaces>& cl =
  basic_clip_props<GraphicsSurfaces>{}) noexcept;

```

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

8 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_draw_text(sfc, bb, b, font, t, tp, bp, sp, d, rp, cl)`.

17.2.6.4 Accessors

[io2d.cmdlists.drawtext.acc]

```

const data_type& data() const noexcept;
data_type& data() noexcept;

```

1 *Returns:* A reference to the `draw_text` object's data object (See: [17.2.6.1](#)).

2 *Remarks:* The behavior of a program is undefined if the user modifies the data contained in the `data_type` object returned by this function.

17.2.6.5 Modifiers

[io2d.cmdlists.drawtext.mod]

```

void surface(
  optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
noexcept;

```

1 *Effects:* Calls `GraphicsSurfaces::surfaces::surface(data(), sfc)`.

2 *Remarks:* The optional surface is `sfc`.

```

void location(const basic_point_2d<graphics_math_type>& pt) noexcept;

```

3 *Effects:* Calls `GraphicsSurfaces::surfaces::location(data(), pt)`.

4 *Remarks:* The text location holds `pt` as its value.

```

void location(const basic_bounding_box<graphics_math_type>& bb) noexcept;
5   Effects: Calls GraphicsSurfaces::surfaces::location(data(), bb).
6   Remarks: The text location holds bb as its value.

void brush(const basic_brush<GraphicsSurfaces>& b) noexcept;
7   Effects: Calls GraphicsSurfaces::surfaces::brush(data(), b).
8   Remarks: The brush is b.

void font(const basic_font<GraphicsSurfaces>& f) noexcept;
9   Effects: Calls GraphicsSurfaces::surfaces::font(data(), f).
10  Remarks: The font is f.

void text(string t) noexcept;
11  Effects: Calls GraphicsSurfaces::surfaces::text(data(), b).
12  Remarks: The text is t.

void text_props(const basic_text_props<GraphicsSurfaces>& tp) noexcept;
13  Effects: Calls GraphicsSurfaces::surfaces::text_props(data(), tp).
14  Remarks: The text props is tp.

void brush_props(const basic_brush_props<GraphicsSurfaces>& bp) noexcept;
15  Effects: Calls GraphicsSurfaces::surfaces::brush_props(data(), bp).
16  Remarks: The brush props is bp.

void brush_props(const basic_stroke_props<GraphicsSurfaces>& sp) noexcept;
17  Effects: Calls GraphicsSurfaces::surfaces::stroke_props(data(), sp).
18  Remarks: The stroke props is sp.

void dashes(const basic_dashes<GraphicsSurfaces>& d) noexcept;
19  Effects: Calls GraphicsSurfaces::surfaces::dashes(data(), d).
20  Remarks: The dashes is d.

void render_props(const basic_render_props<GraphicsSurfaces>& rp) noexcept;
21  Effects: Calls GraphicsSurfaces::surfaces::render_props(data(), rp).
22  Remarks: The render props is rp.

void clip_props(const basic_clip_props<GraphicsSurfaces>& cl) noexcept;
23  Effects: Calls GraphicsSurfaces::surfaces::clip_props(data(), cl).
24  Remarks: The clip props is cl.

```

17.2.6.6 Observers

[io2d.cmdlists.drawtext.obs]

```

optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
  surface() const noexcept;
1   Returns: GraphicsSurfaces::surfaces::surface(data()).
2   Remarks: The returned value is the optional surface.

variant<basic_point_2d<graphics_math_type>, basic_bounding_box<graphics_math_type>> brush() const noexcept;
3   Returns: GraphicsSurfaces::surfaces::location(data()).
4   Remarks: The returned value is the text location.

basic_brush<GraphicsSurfaces> brush() const noexcept;
5   Returns: GraphicsSurfaces::surfaces::brush(data()).
6   Remarks: The returned value is the brush.

```

```
basic_font<GraphicsSurfaces> font() const noexcept;
```

7 *Returns:* GraphicsSurfaces::surfaces::font(data()).

8 *Remarks:* The returned value is the font.

```
basic_text_props<GraphicsSurfaces> text_props() const noexcept;
```

9 *Returns:* GraphicsSurfaces::surfaces::text_props(data()).

10 *Remarks:* The returned value is the text props.

```
basic_brush_props<GraphicsSurfaces> brush_props() const noexcept;
```

11 *Returns:* GraphicsSurfaces::surfaces::brush_props(data()).

12 *Remarks:* The returned value is the brush props.

```
basic_stroke_props<GraphicsSurfaces> stroke_props() const noexcept;
```

13 *Returns:* GraphicsSurfaces::surfaces::stroke_props(data()).

14 *Remarks:* The returned value is the stroke props.

```
basic_dashes<GraphicsSurfaces> dashes() const noexcept;
```

15 *Returns:* GraphicsSurfaces::surfaces::dashes(data()).

16 *Remarks:* The returned value is the dashes.

```
basic_render_props<GraphicsSurfaces> render_props() const noexcept;
```

17 *Returns:* GraphicsSurfaces::surfaces::render_props(data()).

18 *Remarks:* The returned value is the render props.

```
basic_clip_props<GraphicsSurfaces> clip_props() const noexcept;
```

19 *Returns:* GraphicsSurfaces::surfaces::clip_props(data()).

20 *Remarks:* The returned value is the clip props.

17.2.6.7 Equality operators

[io2d.cmdlists.drawtext.eq]

```
template <class GraphicsSurfaces>
```

```
bool operator==(
```

```
    const typename basic_commands<GraphicsSurfaces::draw_text& lhs,
```

```
    const typename basic_commands<GraphicsSurfaces::draw_text& rhs)
```

```
noexcept;
```

1 *Returns:* GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data()).

```
template <class GraphicsSurfaces>
```

```
bool operator!=(
```

```
    const typename basic_commands<GraphicsSurfaces::draw_text& lhs,
```

```
    const typename basic_commands<GraphicsSurfaces::draw_text& rhs)
```

```
noexcept;
```

2 *Returns:* GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data()).

17.2.7 Class template basic_commands<GraphicsSurfaces>::run_function

[io2d.cmdlists.commands.runfunc]

17.2.7.1 Overview

[io2d.cmdlists.runfunc.intro]

1 The class template `basic_commands<GraphicsSurfaces>::run_function` describes a command that invokes the user-provided function, passing it a reference to the surface the command list was submitted to, an optional surface, and user data. It allows the user to perform arbitrary operations that are not otherwise possible using the other command types.

2 It has a *user-provided function* of type `variant<function<void(basic_image_surface<GraphicsSurfaces>&, optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>, void*)>, function<void(basic_output_surface<GraphicsSurfaces>&, optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>, void*)>, function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&, optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>, void*)>>`.

- 3 [Note: The user-defined function is stored in a variant to avoid having three separate classes that essentially provide the same functionality. — end note]
- 4 It has an *optional surface* of type `optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>`.
- 5 It has *user data* of type `void*`.
- 6 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::run_function_data_type`. It is accessible using the *data* member functions.

17.2.7.2 Synopsis

[io2d.cmdlists.runfunc.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_commands<GraphicsSurfaces::run_function {
    public:
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;
        using data_type =
            typename GraphicsSurfaces::surfaces::run_function_data_type;

        // 17.2.7.3, construct:
        run_function(const function<void(basic_image_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>& fn, void* ud,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;
        run_function(const function<void(basic_output_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>& fn, void* ud,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;
        run_function(
            const function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>& fn, void* ud,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;

        // 17.2.7.4, accessors:
        const data_type& data() const noexcept;
        data_type& data() noexcept;

        // 17.2.7.5, modifiers:
        void surface(
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
            noexcept;
        void func(const function<void(basic_image_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>& fn) noexcept;
        void func(const function<void(basic_output_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>& fn) noexcept;
        void func(
            const function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>& fn) noexcept;
        void user_data(void* ud) noexcept;

        // 17.2.7.6, observers:
        optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
            surface() const noexcept;
        const variant<function<void(basic_image_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>, function<void(basic_output_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>, function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
            void*)>>,
            optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>>
```

```

    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*>>& func() const noexcept;
    void* user_data() const noexcept;
};

// 17.2.7.7, equality operators:
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::run_function& lhs,
    const typename basic_commands<GraphicsSurfaces::run_function& rhs)
    noexcept;
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::run_function& lhs,
    const typename basic_commands<GraphicsSurfaces::run_function& rhs)
    noexcept;
}

```

17.2.7.3 Constructors

[io2d.cmdlists.runfunc.ctor]

```

run_function(const function<void(basic_image_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>& fn, void* ud,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;
run_function(const function<void(basic_output_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>& fn, void* ud,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;
run_function(
    const function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>& fn, void* ud,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;

```

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

2 *Postconditions:* `data() == GraphicsSurfaces::surfaces::create_run_function(fn, ud, sfc)`.

17.2.7.4 Accessors

[io2d.cmdlists.runfunc.acc]

```

const data_type& data() const noexcept;
data_type& data() noexcept;

```

1 *Returns:* A reference to the `run_function` object's data object (See: 17.2.7.1).

17.2.7.5 Modifiers

[io2d.cmdlists.runfunc.mod]

```

void surface(
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>> sfc)
    noexcept;

```

1 *Effects:* Calls `GraphicsSurfaces::surfaces::surface(data(), sfc)`.

2 *Remarks:* The optional surface is `sfc`.

```

void func(const function<void(basic_image_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>& fn) noexcept;
void func(const function<void(basic_output_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>& fn) noexcept;

```

```
void func(
    const function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&,
        optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
        void*)>& fn) noexcept;
```

3 *Effects:* Calls `GraphicsSurfaces::surfaces::func(data(), fn)`.

4 *Remarks:* The user-defined function holds `fn` as its value.

```
void user_data(void* ud) noexcept;
```

5 *Effects:* Calls `GraphicsSurfaces::surfaces::user_data(data(), ud)`.

6 *Remarks:* The user data is `ud`.

17.2.7.6 Observers

[io2d.cmdlists.runfunc.obs]

```
optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>
    surface() const noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::surface(data())`.

2 *Remarks:* The returned value is the optional surface.

```
const variant<function<void(basic_image_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>, function<void(basic_output_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>, function<void(basic_unmanaged_output_surface<GraphicsSurfaces>&,
    optional<reference_wrapper<basic_image_surface<GraphicsSurfaces>>>,
    void*)>>& func() const noexcept;
```

3 *Returns:* `GraphicsSurfaces::surfaces::func(data())`.

4 *Remarks:* The returned value is the user-defined function.

```
void* user_data() const noexcept;
```

5 *Returns:* `GraphicsSurfaces::surfaces::user_data(data())`.

6 *Remarks:* The returned value is the user data.

17.2.7.7 Equality operators

[io2d.cmdlists.runfunc.eq]

```
template <class GraphicsSurfaces>
bool operator==(
    const typename basic_commands<GraphicsSurfaces::run_function& lhs,
    const typename basic_commands<GraphicsSurfaces::run_function& rhs)
    noexcept;
```

1 *Returns:* `GraphicsSurfaces::surfaces::equal(lhs.data(), rhs.data())`.

```
template <class GraphicsSurfaces>
bool operator!=(
    const typename basic_commands<GraphicsSurfaces::run_function& lhs,
    const typename basic_commands<GraphicsSurfaces::run_function& rhs)
    noexcept;
```

2 *Returns:* `GraphicsSurfaces::surfaces::not_equal(lhs.data(), rhs.data())`.

17.3 Class template `basic_command_list`

[io2d.commandlist]

17.3.1 Overview

[io2d.commandlist.intro]

1 The class template `basic_command_list<GraphicsSurfaces>` contains the data that results from a back end pre-compiling (interpreting) a sequence of `basic_commands<GraphicsSurfaces>::command_item` objects.

2 This command list may later be executed by one of the surface types. For `basic_image_surface`, it may be executed on separate thread.

3 The data are stored in an object of type `typename GraphicsSurfaces::surfaces::command_list_data_type`. It is accessible using the `data` member function.

17.3.2 basic_command_list synopsis**[io2d.commandlist.synopsis]**

```

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    class basic_command_list {
    public:
        using data_type = typename
            GraphicsSurfaces::surfaces::command_list_data_type;

        // 17.3.3, construct:
        basic_command_list() noexcept;
        template <class InputIterator>
        basic_command_list(InputIterator first, InputIterator last);
        explicit basic_command_list(initializer_list<typename
            basic_commands<GraphicsSurfaces>::command_item>> il);

        // 17.3.4, accessors:
        const data_type& data() const noexcept;
    };
}

```

17.3.3 basic_command_list constructors**[io2d.commandlist.ctor]**

```

basic_command_list() noexcept;
2     Effects: Constructs an object of type basic_command_list.
3     Postconditions: data() == GraphicsSurfaces::surfaces::create_command_list().

explicit basic_command_list(const basic_bounding_box<GraphicsMath>& bb);
4     Effects: Constructs an object of type basic_command_list.
5     Postconditions: data() == GraphicsSurfaces::surfaces::create_command_list().

template <class InputIterator>
basic_command_list(InputIterator first, InputIterator last);
6     Effects: Constructs an object of type basic_command_list.
7     Postconditions: data() == GraphicsSurfaces::surfaces::create_command_list(first, last).
8     [Note: The contained data is the result of the back end pre-compiling the series of objects of type
    basic_commands<GraphicsSurfaces>::command_item from first to the last element before last.
    — end note]

explicit basic_command_list(initializer_list<typename
    basic_commands<GraphicsSurfaces>::command_item> il);
9     Effects: Equivalent to: basic_command_list{ il.begin(), il.end() }.

```

17.3.4 Accessors**[io2d.commandlist.acc]**

```

const data_type& data() const noexcept;
1     Returns: A reference to the basic_command_list object's data object (See: 17.3.1).

namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    struct basic_commands {
        using graphics_math_type = typename GraphicsSurfaces::graphics_math_type;

        class clear;
        class paint;
        class stroke;
        class fill;
        class mask;
        class draw_text;
        class run_function;
    }
}

```



```
using command_item = variant<clear, paint, stroke, fill, mask, draw_text,  
    run_function>;  
};
```

18 Input

[io2d.input]

- ¹ [*Note:* Input, such as keyboard, mouse, and touch, to user-visible surfaces will be added at a later date. This section is a placeholder. It is expected that input will be added via deriving from a user-visible surface. One example is a `basic_io_surface` class template deriving from `basic_output_surface`. This would allow developers to choose not to incur any additional costs of input support where the surface does not require user input. — *end note*]

19 Standalone functions [io2d.standalone]

19.1 Standalone functions synopsis [io2d.standalone.synopsis]

```
namespace std::experimental::io2d::v1 {
    template <class GraphicsSurfaces>
    basic_image_surface<GraphicsSurfaces> copy_surface(
        basic_image_surface<GraphicsSurfaces>& sfc) noexcept;
    template <class GraphicsSurfaces>
    basic_image_surface<GraphicsSurfaces> copy_surface(
        basic_output_surface<GraphicsSurfaces>& sfc) noexcept;
    template <class T>
    constexpr T degrees_to_radians(T d) noexcept;
    template <class T>
    constexpr T radians_to_degrees(T r) noexcept;
    float angle_for_point(point_2d ctr, point_2d pt) noexcept;
    point_2d point_for_angle(float ang, float rad = 1.0f) noexcept;
    point_2d point_for_angle(float ang, point_2d rad) noexcept;
    point_2d arc_start(point_2d ctr, float sang, point_2d rad,
        const matrix_2d& m = matrix_2d{}) noexcept;
    point_2d arc_center(point_2d cpt, float sang, point_2d rad,
        const matrix_2d& m = matrix_2d{}) noexcept;
    point_2d arc_end(point_2d cpt, float eang, point_2d rad,
        const matrix_2d& m = matrix_2d{}) noexcept;
}
```

19.2 copy_surface [io2d.standalone.copysurface]

```
template <class GraphicsSurfaces>
basic_image_surface<GraphicsSurfaces> copy_surface(
    basic_image_surface<GraphicsSurfaces>& sfc) noexcept;
template <class GraphicsSurfaces>
basic_image_surface<GraphicsSurfaces> copy_surface(
    basic_output_surface<GraphicsSurfaces>& sfc) noexcept;
```

¹ *Returns:* GraphicsSurfaces::surfaces::copy_surface(sfc).

19.3 degrees_to_radians [io2d.standalone.degtorad]

```
template <class T>
constexpr T degrees_to_radians(T d) noexcept;
```

Returns: If *d* is positive and is less than one thousandth of a degree, then `static_cast<T>(0)`. If *d* is negative and is less than one thousandth of a degree, then `-static_cast<T>(0)`. Otherwise, the value obtained from converting the degrees value *d* to radians.

Remarks: This function shall not participate in overload resolution unless *T* is a floating-point type.

19.4 radians_to_degrees [io2d.standalone.radtodeg]

```
template <class T>
constexpr T radians_to_degrees(T r) noexcept;
```

Returns: If *r* is positive and is less than one thousandth of a degree in radians, then `static_cast<T>(0)`. If *r* is negative and is less than one thousandth of a degree in radians, then `-static_cast<T>(0)`. Otherwise, the value obtained from converting the radians value *r* to degrees.

Remarks: This function shall not participate in overload resolution unless *T* is a floating-point type.

19.5 angle_for_point [io2d.standalone.angleforpoint]

```
float angle_for_point(point_2d ctr, point_2d pt) noexcept;
```

¹ *Returns:* The angle, in radians, of *pt* as a point on a circle with a center at *ctr*. If the angle is less than `pi<float> / 180000.0f`, returns `0.0f`.

19.6 point_for_angle**[io2d.standalone.pointforangle]**

```
point_2d point_for_angle(float ang, float rad = 1.0f) noexcept;
point_2d point_for_angle(float ang, point_2d rad) noexcept;
```

1 *Requires:* If it is a float, rad is greater than 0.0f. If it is a point_2d, rad.x or rad.y is greater than 0.0f and neither is less than 0.0f.

2 *Returns:* The result of rotating the point point_2d{ 1.0f, 0.0f }, around an origin of point_2d{ 0.0f, 0.0f } by ang radians, with a positive value of ang meaning counterclockwise rotation and a negative value meaning clockwise rotation, with the result being multiplied by rad.

19.7 arc_start**[io2d.standalone.arcstart]**

```
point_2d arc_start(point_2d ctr, float sang, point_2d rad,
  const matrix_2d& m = matrix_2d{}) noexcept;
```

1 *Requires:* rad.x and rad.y are both greater than 0.0f.

2 *Returns:* As-if:

```
auto lmtx = m;
lmtx.m20 = 0.0f; lmtx.m21 = 0.0f;
auto pt = point_for_angle(sang, rad);
return ctr + pt * lmtx;
```

3 [*Note:* Among other things, this function is useful for determining the point at which a new figure should begin if the first item in the figure is an arc and the user wishes to clearly define its center. — *end note*]

19.8 arc_center**[io2d.standalone.arccenter]**

```
point_2d arc_center(point_2d cpt, float sang, point_2d rad,
  const matrix_2d& m = matrix_2d{}) noexcept;
```

1 *Requires:* rad.x and rad.y are both greater than 0.0f.

2 *Returns:* As-if:

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

19.9 arc_end**[io2d.standalone.arcend]**

```
point_2d arc_end(point_2d cpt, float eang, point_2d rad,
  const matrix_2d& m = matrix_2d{}) noexcept;
```

1 *Requires:* rad.x and rad.y are both greater than 0.0f.

2 *Returns:* As-if:

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

Annex A (informative)

Bibliography

[bibliography]

¹ The following is a list of informative resources intended to assist in the understanding or use of this Technical Specification .

- (1.1) — Porter, Thomas and Duff, Tom, 1984, Compositing digital images. ACM SIGGRAPH Computer Graphics. 1984. Vol. 18, no. 3, p. 253-259. DOI 10.1145/964965.808606. Association for Computing Machinery (ACM)
- (1.2) — Foley, James D. et al., *Computer graphics: principles and practice*. 2nd ed. Reading, Massachusetts : Addison-Wesley, 1996.

Annex B (informative)

Response to P1062R0

[p1062r0]

B.1 Overview of P1062R0

[p1062r0.overview]

- ¹ P1062R0 examines and critiques P0267R7. The authors are of the opinion that "[P0267r7], "A Proposal to Add 2D Graphics Rendering and Display to C++" is not a good fit for C++."
- ² We respectfully disagree for a variety of reasons, some of which will be set forth below.
- ³ That said, we offer our thanks to the authors of P1062R0 because they did point out some issues that needed to be addressed and features that should have been (and now are) part of the proposed API.
- ⁴ We will address the various substantive parts of P1062R0 below. The order they are addressed is not the same as their order in the paper for various reasons.
- ⁵ To avoid needless repetition, whenever a section number is given without naming a source document, the section number refers to the corresponding section in P1062R0.

B.2 Section 4

[p1062r0.sec4]

B.2.1 Lack of batching

[p1062r0.sec4.batching]

- ¹ Section 4.1 points out that the API design presented in P0267R7 lacked a batching API that would allow graphical operations such as *paint* and *fill* to be sent in bulk to a hardware accelerated implementation.
- ² While it should be possible to batch all of the drawing commands contained in a *draw callback* and only submit them when an operation with observable behavior is invoked, having a batching API is nonetheless valuable both for hardware accelerated back ends and for other use cases such as specifying a set of operations as data (as opposed to as expressions within a function), applying the same set of operations to several surfaces, and using algorithms to modify the operations dynamically as the program executes in order to more easily achieve various desired results.
- ³ This valuable feedback resulted in the addition of `command_lists` in P0267R9 (See: [Clause 17](#)).

B.2.2 Text and Unicode

[p1062r0.sec4.text]

- ¹ Section 4.2 notes that in R7, there was a Clause on text rendering and display that consisted solely of a placeholder note stating that: "Text rendering and matters related to it, such as font support, will be added at a later date. This section is a placeholder. The integration of text rendering is expected to result in the addition of member functions to the surface class and changes to other parts of the text."
- ² Section 4.2 goes on to state: "[w]hy is this feature missing from the library? This component of the library would depend on the standardization of a text library with Unicode support. There is work ongoing in this space, but it has not yet reached maturity. How much value and utility is there in producing a drawing library that cannot draw text?"
- ³ This placeholder section was added as a result of LEWG feedback on P0267R6 at the Toronto 2017 meeting to denote that text rendering would be part of the final product. Because text rendering is a non-trivial feature, the placeholder text stated that the functionality of drawing text would be added at a later date.
- ⁴ With the resolution of other important design issues in R8, there was sufficient time to add a text rendering API to R9. This API leverages two other standards, ISO/IEC 10646 (what is effectively the ISO/IEC standardization of Unicode) and ISO/IEC 14496-22 (Open Font Format). Together, they encompass the expertise of subject matter experts in those respective fields, providing the bulk of the information necessary to perform text rendering.
- ⁵ What remained was the task of specifying a rendering API and associated data types that provide the information necessary to utilize the desired functionality provided by those standards. This involved becoming familiar with both standards, especially ISO/IEC 14496-22, which is in excess of 600 pages. The combination of waiting until the other parts of the drawing API stabilized in terms of their design and then finding the time to gain that familiarity was the primary factor in text rendering being the last major feature of the output API to be designed and specified.

- ⁶ [*Note*: The text rendering API was reviewed in Cologne by SG16 (Unicode) and no major concerns with it were raised. — *end note*]

B.2.3 Geometry and linear algebra types [p1062r0.sec4.math]

- ¹ Section 4.3 is concerned with the linear algebra and geometry types contained in the proposal.
- ² "These abstractions are far more general than 2D drawing. Why are they being designed in this paper, and why are they being designed specifically for 2D drawing?"
- ³ The reason for this is that they are needed for 2D graphics and this proposal aims to provide 2D graphics library, not a generalized linear algebra and geometry library.
- ⁴ This question has been asked several times over the past several years and each time the answer has been that the authors invite and would very much welcome a proposal for a standardized linear algebra and/or geometry library and would gladly incorporate it into this proposal if and when it was adopted into the Standard.
- ⁵ The paper also raised concerns about the use of `float` rather than a parameterized type. The initial design used `double`. The issue was raised at one meeting that this would provide severe performance issues for hardware accelerated back ends and that `float` was the better choice. The suggestion of having a parameterized types was raised but it did not gain any traction. Similarly, the idea of delaying the paper to wait on a library that might be adopted and shipped sometime in the future was dismissed without anyone feeling the need for any discussion.
- ⁶ Work is now underway on efforts to create a standardized linear algebra library. P1385 is steadily evolving. This is a very welcome development. That said, the earliest it would become part of the standard would be four years from now.
- ⁷ This proposal aims to ship as a TS. The narrowly tailored linear algebra and geometry types live within the `std::experimental::io2d::v1` namespace (n.b. `v1` is an inline namespace). They have been reviewed many times, feedback has been incorporated, and the authors believe that the types are sufficiently ready to be tested by the broader public.
- ⁸ [*Note*: While there is no `operator[] []`, it can be made to seem like there is by using a proxy type. As such, the `basic_matrix` object's `operator[]` needs to create an object of that proxy type, call the proxy object's `operator[]` and then return the returned value. This adds a certain amount of complexity. Given that users normally shouldn't need to access the elements individually, we believe that keeping the existing `mXY` accessor functions design is preferable. — *end note*]

B.2.4 New container type [p1062r0.sec4.pathbuilder]

- ¹ The addition of the container type `path_builder` is objected to in section 4.4.
- ² The existence (and name) of this type has come up in past meetings. Its existence was sustained and its name was changed from `path_factory` to `path_builder`. It is a convenience type that vastly simplifies creating paths, especially for new programmers. Having consulted with maintainers of major standard library implementations, adding and maintaining an additional container type was deemed trivial.

B.3 Section 5 [p1062r0.sec5]

- ¹ Section 5 delves into priorities, committee time limitations, what belongs in the Standard, and related matters.
- ² P0267 began life as N3888. It has been reviewed by LEWG in ordinary sessions at numerous committee meetings over the years. At every such meeting, when the straw poll of whether LEWG wanted to see the proposal again was asked, LEWG decided in favor of it. Indeed, in Kona 2017 the following LEWG straw poll was held:
- ³ "Are we comfortable moving the paper to LWG for a Graphics TS once these concerns are addressed and were happy with the wording."
- ⁴ The vote was 1 strongly in favor, 8 in favor, 3 neutral, 1 against, and 1 strongly against. This was deemed consensus.
- ⁵ The only time there was a vote against continuing P0267 was in the evening session in Rapperswil 2018, a meeting that the primary author did not attend because there was no new revision of P0267 to present and evening sessions had never before been anything other than informational.

⁶ In response, a number National Body chairs co-authored a paper, P1200, stating their unwavering support for P0267 moving forward and becoming a TS. As a result, P0267 has been revived and is presently primarily being considered by SG13.

⁷ As such, we feel that further discussion about the issues raised in section 5 would best be addressed as a reply to P1200.

B.4 Section 3

[p1062r0.sec3]

¹ Section 3 purports to discuss the utility of this proposal both for teaching and for use in applications programming.

² In addition to reviving P0267, P1200 also discussed the goals its authors believed it served, what goals they believed it was not meant to serve, and their rebuttals of various arguments raised against P0267. Those arguments address, broadly, many of the points raised in section 3.

³ There are several issues with section 3 that cannot go without comment, however.

⁴ Section 3.1 states that "[a] simple facility for building graphical interfaces that supports and leverages established graphics standards and formats would allow programmers to use common image editing tools to generate art assets instead of having to express them programmatically in C++".

⁵ We agree. That's why it has been part of the API since P0267R0, the first revision of the proposal to contain formal API wording, and was present conceptually beginning with N3888, the initial paper that began the proposal (the proposal became P0267 once the P number system was adopted).

⁶ Section 3.2 also discusses how programmers would wish to use art assets created by external tools or produced by artists, etc. Again, this is perfectly possible.

⁷ While the number of image formats that back ends are required to support is limited, this is because of complications related to other popular formats not being formally standardized or being otherwise encumbered by restrictions that should not be forced upon back end developers.

⁸ Back end implementations are free to support additional asset formats and are encouraged to do so. There is even a mechanism reserving various enumerators to them for exactly this purpose. There are many things that we cannot or do not require via standardization that implementations provide for the convenience of their users. It is highly unlikely that this would be any different.

⁹ On the whole, section 3 could be read to imply that P0267 does not support drawing image assets and instead only supports programmatically-generated line art. The proposal supports both and has done so from the very beginning.

B.5 Final thoughts and comments

[p1062r0.dtor]

¹ Work is underway on the input API for P0267. While the output API is already very useful, it has always been the intention that this be an I/O API.

² As stated elsewhere in this proposal, from the outset Standard C++ had the functionality to let programs engage with users via the dominant interactive I/O method. At the time that was the console. The dominant interactive I/O method is now the output of 2D graphics and user input via keyboard, mouse, or touch in response to that output. This proposal, amongst the many other things it will do, will restore that functionality to C++.

Annex C (informative)

Response to P1225R0

[p1225r0]

C.1 Overview of P1225R0

[p1225r0.overview]

- ¹ P1225R0 provides feedback on P0267R8 and opinions on what a 2D graphics library should contain.
- ² We offer our thanks for the feedback and our replies to it below.
- ³ To avoid needless repetition, whenever a section number is given without naming a source document, the section number refers to the corresponding section in P1225R0.

C.2 Design

[p1225r0.sec2]

- ¹ We'll address each point in section 2 in turn:
 - (1.1) — 1. Multiple output devices. The API currently support memory buffers (`basic_image_surface`) and windows (`basic_output_surface` and `basic_unmanaged_output_surface`). Additionally, while the reference implementation does not currently contain this functionality, `basic_image_surface` can be implemented in a way that it would record all commands issued to it, thereby allowing it to be output to SVG, PDF, etc., in a way that translates the APIs drawing commands into the closest equivalents in those formats rather than as a raster image. Such an implementation would not preclude using that `basic_image_surface` as a raster image since it could execute the commands and produce the appropriate raster image data on an as-needed basis. As for formats like FP16, we can only support what exists in a published standard since a TS is based against such a document, but we are glad to see progress on additional floating point types and on fixed point types in various proposals. Alpha channel support is present in `basic_image_surface`. It does not exist in the two output surface types because they rarely support alpha (only in certain windowing systems where one window overlays another) and in all such environments we are aware of that do, the host environment controls whether and to what degree the output surface is transparent.
 - (1.2) — 2. Anti-aliasing should come free where supported. This had been the intention all along. It unfortunately got lost somewhere along the way. The paper has been revised to correct this oversight. Thanks!
 - (1.3) — 3. Text. There was placeholder text in R8. A text API was added in R9. The API is not yet complete, but basic functionality is now there.
 - (1.4) — 4. Consistent, DPI-independent, output. We would like to do this but we want to make sure we get it correct. Functionality to enable this might be added to the next revision but we make no promises.
 - (1.5) — 5. Hardware support where available. This is a QoI issue. The current design is that there is a set of class templates that are the "front end". This is what users will use (likely through typedefs to avoid needing to write out `basic_...<SomeImpl>` all the time). The front end interface calls in to a back end, which can be provided by anyone so long as it meets the requirements specified for a `GraphicsSurfaces` type (and the math back end meets the requirements for a `GraphicsMath` type). The back end is specified as a series of typedef'd opaque data structures that are used by static member functions specified by the requirements. A back end can be written to use any graphics technology and support any platform that the author wants. The reference implementation provides three back ends currently (cairo, CoreGraphics, and Qt). We expect many others will be written both before and after the proposal becomes a TS.
 - (1.6) — 6. Reasonable performance. This is also a QoI issue.
 - (1.7) — 7. Reasonable power consumption. This is also a QoI issue.
 - (1.8) — 8. Color spaces and gamma support. The proposal describes sRGB and color spaces in general along with gamma. It does not forbid implementations from making use of any particular color space, but there is no API to access or modify this. We think it's a QoI issue but we welcome suggestions for an API that would expose these in a portable way.

- (1.9) — 9. Possibility to build an interactive model with animation on top of the API. This is possible. See, e.g., the `rocks_in_space` sample for some basic examples of think the matrix modification features in the path API, and the transformation matrices for surfaces and surface state objects.

C.3 Science and teaching [p1225r0.sec3]

- ¹ Regarding section 3, for various reasons stated in this papers and others about it, we think (as do the authors of, e.g., P1200) that this will be a useful teaching tool.
- ² As for scientific plot generation, this paper is the first time we (or the primary author at least) has heard of any such demand. The library is, by intent, fairly low level. We wish to avoid feature creep and anything approaching a GUI-level API. Nonetheless, if a reasonable case is made for adding this specific functionality rather than letting users do it with the existing path API, we're happy to entertain it.

C.4 Abstraction level [p1225r0.sec4]

- ¹ Of the items in section 4:
- (1.1) — Window objects can be obtained if the back end supports it (these data can be obtained from the implementation's `surface_data_type` objects, but this will likely not be cross-platform and, absent any guarantees from implementations, any attempt to use that data dives into the realm of undefined behavior since the contents of the back end `*_data_type` objects are intentionally unspecified).
- (1.2) — Asset file operations are fully supported.
- (1.3) — New user-implemented rasterization primitives are supported to the extent that they can be implemented with the existing API (of the examples given an ellipse can be created using the path API's `arc` functionality while NURBS can only be approximated using the path API's cubic Bézier curves; we're unaware of any widely used cross-platform 2D graphics library that supports them and requiring back end implementers to support them thus would seem to be asking a bit too much perhaps).
- (1.4) — Stacking geometric transforms is possible (there are a number of matrices that are available which will modify the results of the various drawing operations).
- (1.5) — Scissoring and clipping are both available (both via the `basic_clipping` type and by performing a mask operation).
- (1.6) — User input is the next (and final) major feature we will be adding. We intentionally decided to begin with output only for a variety of reasons. Primarily because tackling both at the same time seemed like it would be too big of a task and input doesn't make sense if it's not in response to some sort of output.
- (1.7) — Text support was added in the R9 revision, one after the R8 revision that this paper is critiquing. The API is a work in progress, as described earlier.
- (1.8) — Complex line drawing is possible, including custom dash patterns.
- (1.9) — We believe that the API can be implemented entirely using hardware acceleration, including using shaders where appropriate. (If anyone wishes to engage the primary offer with a contract to create such an implementation... :)).

C.5 Missing details [p1225r0.sec5]

- ¹ We will only briefly touch on section 5 since much of it has been addressed above.
- ² R9 added a command list type that enables buffering of commands and a subsequent bulk execution.
- ³ The image types that are guaranteed to be supported are set out in the `image_file_format` enum class. We wish we could guarantee more, but other popular image formats are either not standardized or are encumbered in various ways which we do not want to force on back end implementers. Back end implementers are free to support more image formats than the guaranteed ones and there is even a specification for how back ends can validly supply their own additional enumerators to indicate that they support various image types beyond the required ones.
- ⁴ To fully support the PDF format (which is standardized), the implementation needs one more brush type, one that would support the PDF Type 7 Shading Type (tensor-product patch meshes). We have not had the time to address that and welcome offers to contribution wording to add that support. We may also need some additional text functionality; this is part of what we are reviewing in terms of expanding the text API that was introduced in R9. Once we have those things we can add PDF support officially. Until then it's something that back ends can choose to support (though they would need to either add that additional

functionality or else only partially support PDFs, which is not an option for us because of the licensing issues that go along with the PDF standard).

C.6 C++ aesthetics and Conclusion

[p1225r0.sec6and7]

- ¹ Lastly lets delve in to sections 6 and 7.
- ² We also dislike the dual error handling mechanism. We have tried to limit it as much as possible and may eliminate it altogether. But we are also hoping that some proposals that would eliminate this issue will gain traction and enter into the Standard.
- ³ It's perfectly possible to iterate over a path. Until such time as it is transformed into a `basic_interpreted_path` object since that is an intentionally opaque type that is meant to give back end implementers the ability to create optimized data that best suits their target environment(s). Even then, the path data collection used to create the `basic_interpreted_path` object remains and can be transformed and resubmitted for future drawing operations (this is one mechanism to use for animation).
- ⁴ We agree about seeing linear algebra, et al., standardized separately. The goal of this proposal is 2D graphics. We created the minimal mathematical types needed for it. Should a linear algebra library become part of the standard we would strongly support integrating it into this proposal at that time and (likely) deprecating the types that we are currently providing.
- ⁵ There is a CoreGraphics back end that supports both Mac and iOS. It was written and offered to us by Michael Kazakov, who undertook to write it on his own and then contacted us to offer it. He has since become a coauthor.
- ⁶ Thanks for your feedback!

Index

- 2D graphics
 - synopsis, 17–23
- additive color, 10
- aliasing, 13
- alpha channel, 9
- anti-aliasing, 13
- aspect ratio, 9
- Bézier curve, 11
 - cubic, 11
 - quadratic, 11
- C
 - Unicode TR, 8
- channel, 9
- closed figure, 11
- color
 - transparent black, 160
- color channel, 9
- color model, 10
 - RGB, 10
 - RGBA, 10
- color space, 10
 - sRGB, 10
- command, 11, 12
- compose, 12
- composition algorithm, 12
- control point, 11
- CSS Colors Specification, 8
- CSS Fonts Specification, 8
- current point, 11
- definitions, 9–13
- degenerate figure, 12
- degenerate segment, 11
- end point, 10
- figure, 12
- figure item, 12
- filter, 12
- final segment, 11
- format
 - JPEG, 8
 - PNG, 8
 - TIFF, 8
- graphics state data, 12
- initial segment, 11
- integral point, 9
- new figure point, 11
- normalize, 9
- open figure, 11
- Open Font Format, 8
- origin, 9
- output device, 185
- path, 12
- path command, 12
- path transformation matrix, 12
- pixel, 10
- point, 9
- premultiplied format, 10
- raster graphics data, 10
- references
 - normative, 8
- render, 12
- rendering and composing operation, 12
- sample, 12
- scope, 7
- segment, 11
- standard coordinate space, 9
- start point, 10
- surface, 181
- surface class templates, 181
- SVG 1.1 Standard, 8
- UTF-8, 8
- vector graphics data, 10
- visual data, 9
- visual data element, 9
- visual data format, 10

Index of library names

- a
 - rgba_color, 29
- abs_cubic_curve, 122
 - constructor, 123
 - control_pt1, 123
 - control_pt2, 123
 - data, 123
 - end_pt, 123
 - operator==, 124
- abs_line, 126
 - constructor, 127
 - data, 127
 - operator==, 127
 - to, 127
- abs_matrix, 118
 - constructor, 119
 - data, 119
 - matrix, 119
 - operator==, 119
- abs_new_figure, 114
 - at, 115
 - constructor, 114, 115
 - data, 115
 - operator==, 115
- abs_quadratic_curve, 129
 - constructor, 130
 - control_pt, 130
 - data, 130
 - end_pt, 130
 - operator==, 130
- angle_for_point, 237
- angular_direction
 - basic_point_2d, 86
- antialias, 155
- antialiasing
 - basic_fill_props, 175
 - basic_stroke_props, 173, 174
 - basic_text_props, 104
- arc, 132
 - basic_path_builder, 142
 - center, 134
 - constructor, 133, 134
 - data, 134
 - operator==, 135
 - radius, 134
 - rotation, 134
 - start_angle, 134
- arc_center, 238
- arc_end, 238
- arc_start, 238
- at
 - abs_new_figure, 115
 - rel_new_figure, 116, 117
- auto_clear
 - basic_output_surface, 202, 203
 - basic_unmanaged_output_surface, 206, 208
- b
 - rgba_color, 29
- basic_bounding_box, 93
 - bottom_right, 95
 - constructor, 94
 - data, 94
 - height, 95
 - operator==, 95
 - top_left, 95
 - width, 95
 - x, 94, 95
 - y, 95
- basic_brush, 151
 - constructor, 153, 154
 - data, 154, 177
 - type, 154
- basic_brush_props, 168
 - brush_matrix, 169
 - constructor, 168
 - data, 168
 - filter, 169
 - wrap_mode, 168, 169
- basic_circle, 95
 - center, 96
 - constructor, 96
 - data, 96
 - operator==, 97
 - radius, 96
- basic_clip_props
 - clip, 171, 172
 - constructor, 170, 171
 - data, 171
 - fill_rule, 171, 172
- basic_command_list, 233
 - constructor, 234
 - data, 234
- basic_command_list>
 - command_list, 201
- basic_dashes, 176
 - constructor, 177
- basic_display_point, 92
 - constructor, 92, 93
 - data, 93
 - operator==, 93
 - x, 93
 - y, 93
- basic_fill_props
 - antialiasing, 175
 - constructor, 174

- data, 174
- fill_rule, 175
- filtering, 175
- basic_font, 105
 - constructor, 106
 - data, 107
 - family, 107
 - font_size, 107
 - kerning, 107
 - merging, 107
 - size_units, 107
 - style, 107
 - weight, 107
- basic_font_database, 108
 - constructor, 108
 - data, 108
 - get_families, 108
- basic_image_surface, 190
 - clear, 193
 - command_list, 196
 - constructor, 191
 - data, 192
 - dimensions, 193
 - draw_text, 195
 - fill, 194
 - format, 193
 - mask, 195
 - max_dimensions, 193
 - paint, 193
 - save, 192
 - stroke, 193, 194
- basic_interpreted_path, 136
 - constructor, 137
 - data, 137
- basic_mask_props
 - constructor, 176
 - data, 176
 - filter, 176
 - mask_matrix, 176
 - wrap_mode, 176
- basic_matrix_2d, 87
 - constructor, 89
 - create_reflect, 89
 - create_rotate, 89
 - create_scale, 89
 - create_shear_y, 89, 90
 - create_translate, 89
 - data, 89
 - determinant, 91
 - inverse, 91
 - is_finite, 91
 - is_invertible, 91
 - m00, 90
 - m01, 90, 91
 - m10, 90, 91
 - m11, 90, 91
 - m20, 90, 91
 - m21, 90, 91
 - operator*, 91
 - operator**, 91
 - operator==, 91
 - reflect, 90
 - rotate, 90
 - scale, 90
 - shear_x, 90
 - shear_y, 90
 - transform_pt, 91
 - translate, 90
- basic_output_surface, 196
 - auto_clear, 202, 203
 - begin_show, 199
 - clear, 200
 - constructor, 198, 199
 - dimensions, 202, 203
 - display_dimensions, 202, 203
 - draw_callback, 202
 - draw_text, 201
 - end_show, 200
 - fill, 201
 - format, 203
 - letterbox_brush, 202, 203
 - letterbox_brush_props, 203
 - mask, 201
 - max_dimensions, 203
 - max_output_dimensions, 203
 - paint, 200
 - redraw_required, 202
 - scaling, 202, 203
 - size_change_callback, 202
 - stroke, 200
 - user_scaling_callback, 202
- basic_path_builder, 141
- basic_path_builder, 137
 - arc, 142
 - begin, 143
 - capacity, 140
 - cbegin, 143
 - cend, 143
 - close_figure, 141
 - constructor, 140
 - crbegin, 143
 - crend, 143
 - cubic_curve, 142
 - emplace_back, 142
 - end, 143
 - erase, 143
 - insert, 142
 - line, 142
 - modify_matrix, 142
 - new_figure, 141
 - pop_back, 143
 - push_back, 142
 - quadratic_curve, 142
 - rbegin, 143
 - rel_cubic_curve, 142
 - rel_line, 142
 - rel_new_figure, 141
 - rel_quadratic_curve, 142

- rend, 143
- reserve, 140
- revert_matrix, 142
- set_matrix, 142
- shrink_to_fit, 141
- swap, 141
- basic_point_2d, 84
 - angular_direction, 86
 - constructor, 85
 - data, 85
 - dot, 86
 - magnitude, 86
 - magnitude_squared, 86
 - operator*, 87
 - operator*=: 86
 - operator+, 87
 - operator+=, 86
 - operator-, 87
 - operator-=, 86
 - operator/, 87
 - operator/=, 86
 - operator==, 87
 - to_unit, 86
 - x, 86
 - y, 86
- basic_render_props
 - compositing, 167
 - constructor, 167
 - data, 167
 - filtering, 167
 - surface_matrix, 167
 - surface_member, 167
- basic_stroke_props
 - antialiasing, 173, 174
 - constructor, 172, 173
 - data, 173
 - line_cap, 173
 - line_join, 173
 - line_width, 173
 - max_miter_limit, 174
 - miter_limit, 173
- basic_text_props, 102
 - antialiasing, 104
 - constructor, 103
 - data, 103
 - font_size, 104
 - kerning, 104
 - line, 104, 105
 - scale, 104
 - size_units, 104
 - stretching, 104, 105
 - strike_through, 104, 105
- basic_unmanaged_output_surface, 203
 - auto_clear, 206, 208
 - clear, 206
 - constructor, 205
 - data, 205
 - dimensions, 206, 207
 - display_dimensions, 206, 207
 - draw_callback, 207
 - fill, 207
 - format, 205
 - has_draw_callback, 205
 - has_size_change_callback, 205
 - letterbox_brush, 206, 207
 - letterbox_brush_props, 206
 - mask, 207
 - max_dimensions, 206
 - max_display_dimensions, 206
 - paint, 206
 - scaling, 206, 207
 - size_change_callback, 207
 - stroke, 206
- begin
 - basic_path_builder, 143
- begin_show
 - basic_output_surface, 199
- bottom_right
 - basic_bounding_box, 95
- brush
 - draw_text, 229
 - fill, 220
 - mask, 223, 224
 - paint, 213
 - stroke, 216, 217
- brush_matrix
 - basic_brush_props, 169
- brush_props
 - draw_text, 229, 230
 - fill, 220
 - mask, 223, 224
 - paint, 213
 - stroke, 216, 217
- capacity
 - basic_path_builder, 140
- cbegin
 - basic_path_builder, 143
- cend
 - basic_path_builder, 143
- center
 - arc, 134
 - basic_circle, 96
- clear, 209
 - basic_image_surface, 193
 - basic_output_surface, 200
 - basic_unmanaged_output_surface, 206
 - constructor, 210
 - data, 210
 - operator==, 210
 - surface, 210
- clip
 - basic_clip_props, 171, 172
- clip_props
 - draw_text, 229, 230
 - fill, 220
 - mask, 223, 224
 - paint, 213

- stroke, 217
- close_figure, 117
 - basic_path_builder, 141
 - operator==, 117
- color
 - gradient_stop, 151
- command_list
 - basic_command_list>, 201
 - basic_image_surface, 196
- compositing
 - basic_render_props, 167
- control_pt
 - abs_quadratic_curve, 130
 - rel_quadratic_curve, 132
- control_pt1
 - abs_cubic_curve, 123
 - rel_cubic_curve, 125
- control_pt2
 - abs_cubic_curve, 123
 - rel_cubic_curve, 125
- copy_surface, 237
- crbegin
 - basic_path_builder, 143
- create_reflect
 - basic_matrix_2d, 89
- create_rotate
 - basic_matrix_2d, 89
- create_scale
 - basic_matrix_2d, 89
- create_shear_y
 - basic_matrix_2d, 89, 90
- create_translate
 - basic_matrix_2d, 89
- crend
 - basic_path_builder, 143
- cubic_curve
 - basic_path_builder, 142
- dashes
 - draw_text, 229, 230
 - stroke, 216, 217
- data
 - abs_cubic_curve, 123
 - abs_line, 127
 - abs_matrix, 119
 - abs_new_figure, 115
 - abs_quadratic_curve, 130
 - arc, 134
 - basic_bounding_box, 94
 - basic_brush, 154, 177
 - basic_brush_props, 168
 - basic_circle, 96
 - basic_clip_props, 171
 - basic_command_list, 234
 - basic_display_point, 93
 - basic_fill_props, 174
 - basic_font, 107
 - basic_font_database, 108
 - basic_image_surface, 192
 - basic_interpreted_path, 137
 - basic_mask_props, 176
 - basic_matrix_2d, 89
 - basic_point_2d, 85
 - basic_render_props, 167
 - basic_stroke_props, 173
 - basic_text_props, 103
 - basic_unmanaged_output_surface, 205
 - clear, 210
 - draw_text, 228
 - fill, 219
 - mask, 223
 - paint, 212
 - rel_cubic_curve, 125
 - rel_line, 128
 - rel_matrix, 120
 - rel_new_figure, 116
 - rel_quadratic_curve, 132
 - run_function, 232
 - stroke, 216
- determinant
 - basic_matrix_2d, 91
- dimensions
 - basic_image_surface, 193
 - basic_output_surface, 202, 203
 - basic_unmanaged_output_surface, 206, 207
- display_dimensions
 - basic_output_surface, 202, 203
 - basic_unmanaged_output_surface, 206, 207
- dot
 - basic_point_2d, 86
- draw_callback
 - basic_output_surface, 202
 - basic_unmanaged_output_surface, 207
- draw_text, 224
 - basic_image_surface, 195
 - basic_output_surface, 201
 - brush, 229
 - brush_props, 229, 230
 - clip_props, 229, 230
 - constructor, 227, 228
 - dashes, 229, 230
 - data, 228
 - font, 229, 230
 - location, 228, 229
 - operator==, 230
 - render_props, 229, 230
 - stroke_props, 229, 230
 - surface, 228, 229
 - text, 229
 - text_props, 229, 230
- emplace_back
 - basic_path_builder, 142
- end
 - basic_path_builder, 143
- end_pt

- abs_cubic_curve, 123
- abs_quadratic_curve, 130
- rel_cubic_curve, 125, 126
- rel_quadratic_curve, 132
- end_show
 - basic_output_surface, 200
- erase
 - basic_path_builder, 143
- <experimental/io2d>, 17
- family
 - basic_font, 107
- fill, 218
 - basic_image_surface, 194
 - basic_output_surface, 201
 - basic_unmanaged_output_surface, 207
 - brush, 220
 - brush_props, 220
 - clip_props, 220
 - constructor, 219
 - data, 219
 - operator==, 221
 - path, 220
 - render_props, 220
 - surface, 220
- fill_rule
 - basic_clip_props, 171, 172
 - basic_fill_props, 175
- filter
 - basic_brush_props, 169
 - basic_mask_props, 176
- filtering
 - basic_fill_props, 175
 - basic_render_props, 167
- font
 - draw_text, 229, 230
- font_antialias, 101
- font_capitalization, 99
- font_line, 101
- font_size
 - basic_font, 107
 - basic_text_props, 104
- font_size_units, 98
- font_stretching, 100
- font_style, 100
- font_weight, 99
- format
 - basic_image_surface, 193
 - basic_output_surface, 203
 - basic_unmanaged_output_surface, 205
- func
 - run_function, 232, 233
- g
 - rgba_color, 28, 29
- generic_font_names, 102
- get_families
 - basic_font_database, 108
- gradient_stop, 150
 - color, 151
 - constructor, 151
 - offset, 151
 - operator==, 151
- has_draw_callback
 - basic_unmanaged_output_surface, 205
- has_size_change_callback
 - basic_unmanaged_output_surface, 205
- height
 - basic_bounding_box, 95
- image_file_format, 180
- insert
 - basic_path_builder, 142
- inverse
 - basic_matrix_2d, 91
- is_finite
 - basic_matrix_2d, 91
- is_invertible
 - basic_matrix_2d, 91
- kerning
 - basic_font, 107
 - basic_text_props, 104
- letterbox_brush
 - basic_output_surface, 202, 203
 - basic_unmanaged_output_surface, 206, 207
- letterbox_brush_props
 - basic_output_surface, 203
 - basic_unmanaged_output_surface, 206
- line
 - basic_path_builder, 142
 - basic_text_props, 104, 105
- line_cap
 - basic_stroke_props, 173
- line_join
 - basic_stroke_props, 173
- line_width
 - basic_stroke_props, 173
- location
 - draw_text, 228, 229
- m00
 - basic_matrix_2d, 90
- m01
 - basic_matrix_2d, 90, 91
- m10
 - basic_matrix_2d, 90, 91
- m11
 - basic_matrix_2d, 90, 91
- m20
 - basic_matrix_2d, 90, 91
- m21
 - basic_matrix_2d, 90, 91
- magnitude
 - basic_point_2d, 86
- magnitude_squared

- basic_point_2d, 86
- mask, 221
 - basic_image_surface, 195
 - basic_output_surface, 201
 - basic_unmanaged_output_surface, 207
 - brush, 223, 224
 - brush_props, 223, 224
 - clip_props, 223, 224
 - constructor, 222, 223
 - data, 223
 - mask_brush, 223, 224
 - mask_props, 223, 224
 - operator==, 224
 - render_props, 223, 224
 - surface, 223, 224
- mask_brush
 - mask, 223, 224
- mask_matrix
 - basic_mask_props, 176
- mask_props
 - mask, 223, 224
- matrix
 - abs_matrix, 119
 - rel_matrix, 121
- max_dimensions
 - basic_image_surface, 193
 - basic_output_surface, 203
 - basic_unmanaged_output_surface, 206
- max_display_dimensions
 - basic_unmanaged_output_surface, 206
- max_miter_limit
 - basic_stroke_props, 174
- max_output_dimensions
 - basic_output_surface, 203
- merging
 - basic_font, 107
- miter_limit
 - basic_stroke_props, 173
- modify_matrix
 - basic_path_builder, 142
- new_figure
 - basic_path_builder, 141
- offset
 - gradient_stop, 151
- operator*
 - basic_matrix_2d, 91
 - basic_point_2d, 87
 - rgba_color, 33
- operator*=
 - basic_matrix_2d, 91
 - basic_point_2d, 86
 - rgba_color, 32
- operator+
 - basic_point_2d, 87
- operator+=
 - basic_point_2d, 86
- operator-
 - basic_point_2d, 87
- operator-=
 - basic_point_2d, 86
- operator/
 - basic_point_2d, 87
- operator/=
 - basic_point_2d, 86
- operator==
 - abs_cubic_curve, 124
 - abs_line, 127
 - abs_matrix, 119
 - abs_new_figure, 115
 - abs_quadratic_curve, 130
 - arc, 135
 - basic_bounding_box, 95
 - basic_circle, 97
 - basic_display_point, 93
 - basic_matrix_2d, 91
 - basic_point_2d, 87
 - clear, 210
 - close_figure, 117
 - draw_text, 230
 - fill, 221
 - gradient_stop, 151
 - mask, 224
 - paint, 213
 - rel_cubic_curve, 126
 - rel_line, 129
 - rel_matrix, 121
 - rel_new_figure, 117
 - rel_quadratic_curve, 132
 - revert_matrix, 122
 - rgba_color, 33
 - run_function, 233
 - stroke, 217
- paint, 211
 - basic_image_surface, 193
 - basic_output_surface, 200
 - basic_unmanaged_output_surface, 206
 - brush, 213
 - brush_props, 213
 - clip_props, 213
 - constructor, 212
 - data, 212
 - operator==, 213
 - render_props, 213
 - surface, 212, 213
- path
 - fill, 220
 - stroke, 216, 217
- point_for_angle, 238
- pop_back
 - basic_path_builder, 143
- push_back
 - basic_path_builder, 142
- quadratic_curve
 - basic_path_builder, 142

r
 rgba_color, 28, 29
 radius
 arc, 134
 basic_circle, 96
 rbegin
 basic_path_builder, 143
 redraw_required
 basic_output_surface, 202
 reflect
 basic_matrix_2d, 90
 rel_cubic_curve, 124
 basic_path_builder, 142
 constructor, 125
 control_pt1, 125
 control_pt2, 125
 data, 125
 end_pt, 125, 126
 operator==, 126
 rel_line, 127
 basic_path_builder, 142
 constructor, 128
 data, 128
 operator==, 129
 to, 128, 129
 rel_matrix, 119
 constructor, 120
 data, 120
 matrix, 121
 operator==, 121
 rel_new_figure, 115
 at, 116, 117
 basic_path_builder, 141
 constructor, 116
 data, 116
 operator==, 117
 rel_quadratic_curve, 131
 basic_path_builder, 142
 constructor, 132
 control_pt, 132
 data, 132
 end_pt, 132
 operator==, 132
 rend
 basic_path_builder, 143
 render_props
 draw_text, 229, 230
 fill, 220
 mask, 223, 224
 paint, 213
 stroke, 216, 217
 reserve
 basic_path_builder, 140
 revert_matrix, 121
 basic_path_builder, 142
 operator==, 122
 rgba_color, 24
 a, 29
 b, 29
 constructor, 28
 g, 28, 29
 operator*, 33
 operator==, 32
 operator==, 33
 r, 28, 29
 rotate
 basic_matrix_2d, 90
 rotation
 arc, 134
 run_function, 230
 constructor, 232
 data, 232
 func, 232, 233
 operator==, 233
 surface, 232, 233
 user_data, 233

 save
 basic_image_surface, 192
 scale
 basic_matrix_2d, 90
 basic_text_props, 104
 scaling
 basic_output_surface, 202, 203
 basic_unmanaged_output_surface, 206,
 207
 set_matrix
 basic_path_builder, 142
 shear_x
 basic_matrix_2d, 90
 shear_y
 basic_matrix_2d, 90
 shrink_to_fit
 basic_path_builder, 141
 size_change_callback
 basic_output_surface, 202
 basic_unmanaged_output_surface, 207
 size_units
 basic_font, 107
 basic_text_props, 104
 start_angle
 arc, 134
 stretching
 basic_text_props, 104, 105
 strike_through
 basic_text_props, 104, 105
 stroke, 214
 basic_image_surface, 193, 194
 basic_output_surface, 200
 basic_unmanaged_output_surface, 206
 brush, 216, 217
 brush_props, 216, 217
 clip_props, 217
 constructor, 215, 216
 dashes, 216, 217
 data, 216
 operator==, 217
 path, 216, 217

- render_props, 216, 217
 - stroke_props, 216, 217
 - surface, 216, 217
- stroke_props
 - draw_text, 229, 230
 - stroke, 216, 217
- style
 - basic_font, 107
- surface, 181
 - clear, 210
 - draw_text, 228, 229
 - fill, 220
 - mask, 223, 224
 - paint, 212, 213
 - run_function, 232, 233
 - stroke, 216, 217
- surface_matrix
 - basic_render_props, 167
- surface_member
 - basic_render_props, 167
- swap
 - basic_path_builder, 141, 144
- text
 - draw_text, 229
- text_props
 - draw_text, 229, 230
- to
 - abs_line, 127
 - rel_line, 128, 129
- to_unit
 - basic_point_2d, 86
- top_left
 - basic_bounding_box, 95
- transform_pt
 - basic_matrix_2d, 91
- translate
 - basic_matrix_2d, 90
- user_data
 - run_function, 233
- user_scaling_callback
 - basic_output_surface, 202
- weight
 - basic_font, 107
- width
 - basic_bounding_box, 95
- wrap_mode
 - basic_brush_props, 168, 169
 - basic_mask_props, 176
- x
 - basic_bounding_box, 94, 95
 - basic_display_point, 93
 - basic_point_2d, 86
- y
 - basic_bounding_box, 95
 - basic_display_point, 93
 - basic_point_2d, 86

Index of implementation-defined behavior

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

- additional data formats, [43](#)
- additional read only data formats, [43](#)
- additional visual data formats, [43](#)
- additional write only data formats, [43](#)
- antialiasing
 - antialiasing
 - best, [155](#)
 - antialiasing
 - fast, [155](#)
 - antialiasing
 - good, [155](#)
- basic_image_surface
 - draw_text, [195](#)
 - fill, [194](#), [195](#)
 - mask, [195](#)
 - paint, [193](#)
 - save, [193](#)
 - stroke, [194](#)
- basic_output_surface
 - begin_show, [200](#)
 - begin_show return value, [200](#)
 - constructor, [198](#), [199](#)
 - display_dimensions, [202](#)
 - draw_text, [201](#)
 - fill, [201](#)
 - paint, [200](#)
 - stroke, [200](#)
- basic_unmanaged_output_surface
 - data_type, [205](#)
 - fill, [207](#)
 - mask, [207](#)
 - paint, [206](#)
 - stroke, [207](#)
- create_unmanaged_output_surface, [77](#)
- dash pattern
 - offset value, [184](#)
- filter
 - best, [150](#)
 - fast, [150](#)
 - good, [150](#)
- format::argb32, [163](#)
- format::xrgb16, [163](#)
- format::xrgb32, [163](#)
- implementation system_error-derived type, [14](#)
- max miter limit, [58](#)
- output device, [185](#)
- output device visual format support, [66](#), [69](#)
- output surface
 - unsupported draw format, [187](#)
- surface
 - mask, [201](#)
- type of basic_path_builder::const_iterator, [138](#)
- type of basic_path_builder::iterator, [138](#)
- type of basic_path_builder::size_type, [138](#)
- X::surfaces::create_image_surface, [64](#)
- X::surfaces::save, [64](#)