



Sinneslöschen

SYSTEMS CORPORATION

CLEAR SIGNALS. RELIABLE SYSTEMS.

CUSTOMER SOFTWARE DOCUMENTATION

Graphics Library 4.1

Programmer's Reference

Technical Graphics Programming Library

Document Number:	RM-GL41-01
Revision:	A
Release Date:	June 2, 1998
Software:	Graphics Library 4.1
Platform:	Windows NT 4.0
Distribution:	Customer / OEM / Development

This manual defines the Graphics Library 4.1 C programming interface, including device/context management, surfaces, coordinate transformations, clipping, primitives, text and symbol services, image transfers, synchronization, error handling and VDP-6/VDP-7 hardware acceleration behavior.

Copyright and Technical Notice

Copyright © 1998 Sinneslöschen Systems Corporation. All rights reserved.

The information in this manual is believed accurate as of the publication date. Sinneslöschen Systems Corporation reserves the right to revise software, hardware, documentation and supported configurations without notice. This manual documents Graphics Library 4.1 as supplied for qualified Sinneslöschen technical visualization systems and VDP display hardware.

Graphics Library 4.1 is intended for application developers, OEM engineering groups, laboratory programmers and system integrators. It provides a device-independent programming interface while exposing selected device capabilities when an application requires VDP-specific control.

CAUTION

Graphics Library functions can modify display timing, framebuffer contents, palette state and device configuration indirectly through the installed display driver. Applications used on production or safety-related equipment shall be qualified with the exact library, driver, hardware revision and display configuration in which they will operate.

Technical Support

North American Technical Support: 1-800-555-0142

Direct: 408-555-0186

Fax: 408-555-0127

E-mail: support@sinnesloschen.com

When reporting a library problem, include the Graphics Library version, VDP model and revision, display driver version, operating system service level, pixel format, failing function sequence and the value returned by `sglGetLastError()`.

Document Control

Field	Value
Document	RM-GL41-01 - Graphics Library 4.1 Programmer's Reference
Current Revision	A
Release Date	June 2, 1998
Software	Sinneslösch Graphics Library 4.1
Primary Platform	Windows NT 4.0
Display Hardware	VDP-6, VDP-7, qualified SV visualization systems
Related Software	VDP Development Toolkit 3.2; Technical Font & Symbol Library 3.1
Maintaining Organization	Display Systems Laboratory / System Software / Technical Documentation

Document Purpose

This manual defines the C programming interface, data types, state rules, coordinate model, text and symbol services, image transfer operations, hardware-acceleration behavior and error reporting of Graphics Library 4.1. It is a programming reference, not a tutorial on general computer graphics.

Applicability

Revision A applies to Graphics Library 4.1 build series 4.1.0 through 4.1.3 unless a later release note supersedes a specific statement. Hardware behavior is described only where it affects the library contract; register-level VDP behavior remains defined by the applicable VDP reference manual.

Revision History and Related Publications

Rev.	Date	Status	Summary
A	06.02.98	Current	Initial reference for Graphics Library 4.1. Adds revised display-buffer management, VDP-7 hardware acceleration, corrected clipping/transform behavior and expanded technical font services.

Related Publications

- RM-VDP7-03 - VDP-7 Programmer's Reference Manual, Rev. C.
- RM-VDP6-05 - VDP-6 Programmer's Reference Manual, Rev. E.
- UM-GL41-01 - Graphics Library 4.1 User's Guide.
- AN-VDP7-05 - High-Resolution Display Design with the VDP-7.
- VDP Development Toolkit 3.2 Release Notes.
- Technical Font & Symbol Library 3.1 Reference Notes.

Compatibility Note

Applications written for Graphics Library 4.0 should review Chapter 11 before rebuilding. Version 4.1 retains source compatibility for most documented calls but changes several validation rules, clipping edge cases and buffer-lifetime requirements.

How to Use This Manual

Graphics Library 4.1 can be used at several levels. Simple applications may create one context and draw exclusively to the visible surface. Engineering and OEM programs may maintain multiple off-screen surfaces, perform explicit clipping and transforms, synchronize to vertical blank, select acceleration policies or interoperate with VDP-specific services.

Recommended Reading Paths

- New application development: Chapters 1-7, then the function reference pages used by the application.
- Porting from Graphics Library 4.0: Chapters 2, 3, 10 and 11.
- VDP performance work: Chapters 7, 9 and 10 plus RM-VDP7-03.
- Technical text/symbol applications: Chapters 6 and 8 function references.
- Field diagnosis: Chapter 10, Appendix C and the applicable VDP diagnostic documentation.

API CONTRACT

Only behavior documented in this manual, the matching User's Guide or an applicable release note is considered part of the supported Graphics Library 4.1 interface. Applications shall not depend on undocumented structure members, private entry points, driver-specific environment variables or internal command packet formats.

Conventions, Types and Numeric Formats

Notation / Type	Meaning
SGL_RESULT	Signed result code. Zero indicates success unless a function page states otherwise.
SGL_CONTEXT	Opaque graphics context handle.
SGL_SURFACE	Opaque surface handle.
SGL_DEVICE	Opaque display-device handle.
SGL_COLOR	Logical color value interpreted according to the destination format.
SGL_POINT	Pair of signed 32-bit coordinates.
SGL_RECT	Left, top, right and bottom bounds using library edge conventions.
NULL	Null pointer or invalid optional object as described by the specific call.
SGL_ERR_*	Negative error code returned or recorded by the library.

Unless otherwise stated, coordinates are signed integers expressed in user space and converted through the current transform. Device coordinates are clipped to the bound surface. Rectangle right and bottom edges are exclusive for transfer and fill operations; geometric primitives define their own endpoint rules in Chapter 7.

Source examples use ANSI C syntax accepted by the compiler supplied with Sinneslösch Development Tools 5.3 and Microsoft Visual C++ 5.x in C mode. Header examples assume `#include <sgl41.h>`.

Contents 1 of 6

Section	Subject	Pages
Front Matter	Copyright / Document Control / Conventions	2-6
Contents	Detailed Contents	7-12
Chapter 1	Introduction	13-20
Chapter 2	Installation and Build Environment	21-34
Chapter 3	Programming Model	35-52
Chapter 4	Devices, Contexts and Surfaces	53-70
Chapter 5	Coordinates, Transforms and Clipping	71-90
Chapter 6	Color, Text and Symbols	91-106
Chapter 7	Primitives and Image Operations	107-122
Chapter 8	Function Reference	123-212
Chapter 9	VDP Hardware Acceleration	213-226
Chapter 10	Errors, Threading and Performance	227-238
Chapter 11	Migration and Compatibility	239-248
Chapter 12	Programming Examples	249-258
Appendices	Types, Errors, Capability, Index, Glossary	259-268

Reference Use

Chapter 8 contains one-page function entries intended for desk-side reference. Chapters 3-7 define the state and geometry rules those pages rely upon.

Contents 2 of 6

Page	Topic
13	1.1 Library Overview
14	1.2 Software Stack and Components
15	1.3 Supported Devices and Platforms
16	1.4 Execution Model
17	1.5 Library State
18	1.6 Coordinate and Pixel Independence
19	1.7 Error Model
20	1.8 Minimal Program
21	2.1 Installation Components
22	2.2 Directory Layout
23	2.3 Header Files
24	2.4 Import Libraries and DLLs
25	2.5 Compiler Settings
26	2.6 Building a Console Sample
27	2.7 Building a Win32 Application
28	2.8 Runtime Version Checking
29	2.9 Driver Requirements
30	2.10 Environment and Configuration
31	2.11 Redistributable Runtime
32	2.12 Debug and Release Builds
33	2.13 Installation Verification
34	2.14 Deployment Checklist
35	3.1 Object Model
36	3.2 Device Lifetime
37	3.3 Context Lifetime
38	3.4 Surface Lifetime
39	3.5 Current Context Rules
40	3.6 Draw and Read Surfaces
41	3.7 State Change Ordering
42	3.8 Deferred Execution
43	3.9 Flush and Finish
44	3.10 Begin and End Batch
45	3.11 State Stack
46	3.12 Default State
47	3.13 Capability Queries
48	3.14 Device Reset
49	3.15 Window Resize and Mode Change
50	3.16 Display Ownership
51	3.17 Threading Summary
52	3.18 Determinism and Qualification
53	4.1 Device Enumeration
54	4.2 Device Information
55	4.3 Opening a Device
56	4.4 Closing a Device
57	4.5 Creating a Context
58	4.6 Context Destruction
59	4.7 Front Surface
60	4.8 Back Surface
61	4.9 Off-Screen Surfaces
62	4.10 Surface Descriptors
63	4.11 Surface Binding
64	4.12 Read Surface Selection

Contents 3 of 6

Page	Topic
65	4.13 Surface Locking
66	4.14 Lock Flags
67	4.15 Display Start and Panning
68	4.16 Surface Format Compatibility
69	4.17 Surface Loss
70	4.18 Surface Design Checklist
71	5.1 Coordinate Pipeline
72	5.2 User Coordinates
73	5.3 Identity Mapping
74	5.4 Matrix Representation
75	5.5 Load and Multiply Order
76	5.6 Translation
77	5.7 Scaling
78	5.8 Rotation
79	5.9 Window Mapping
80	5.10 Viewport Mapping
81	5.11 Origin
82	5.12 Clip Rectangle
83	5.13 Clip Enable
84	5.14 Rectangle Edge Rules
85	5.15 Line Endpoint Rules
86	5.16 Subpixel and Integer Paths
87	5.17 Overflow and Range Checking
88	5.18 Mirrored Coordinates
89	5.19 Multi-Panel Layout
90	5.20 Coordinate Checklist
91	6.1 Logical Color Model
92	6.2 Indexed Color
93	6.3 Direct Color Formats
94	6.4 Palette Management
95	6.5 Palette Synchronization
96	6.6 Color Maps
97	6.7 Write Mask
98	6.8 Raster Operations
99	6.9 Font Resources
100	6.10 Technical Font and Symbol Library
101	6.11 Glyph Caching
102	6.12 Text Baseline and Metrics
103	6.13 Text Rotation
104	6.14 Text Clipping
105	6.15 Symbol Rendering
106	6.16 Color and Text Checklist
107	7.1 Primitive Overview
108	7.2 Points
109	7.3 Lines
110	7.4 Polylines
111	7.5 Rectangles
112	7.6 Triangles
113	7.7 Polygons
114	7.8 Circles and Ellipses
115	7.9 Arcs
116	7.10 Fill Patterns

Contents 4 of 6

Page	Topic
117	7.11 Screen-to-Screen Copy
118	7.12 General Blit
119	7.13 Scaled Blit
120	7.14 Host Image Transfer
121	7.15 Clear and Fill
122	7.16 Primitive Qualification Checklist
123	8.1 sglGetVersion
124	8.2 sglEnumerateDevices
125	8.3 sglOpenDevice
126	8.4 sglCloseDevice
127	8.5 sglGetDeviceInfo
128	8.6 sglCreateContext
129	8.7 sglDestroyContext
130	8.8 sglMakeCurrent
131	8.9 sglGetCurrentContext
132	8.10 sglFlush
133	8.11 sglFinish
134	8.12 sglResetContext
135	8.13 sglQueryCapability
136	8.14 sglGetFrontSurface
137	8.15 sglCreateSurface
138	8.16 sglDestroySurface
139	8.17 sglGetSurfaceInfo
140	8.18 sglBindSurface
141	8.19 sglSetReadSurface
142	8.20 sglLockSurface
143	8.21 sglUnlockSurface
144	8.22 sglSetDrawBuffer
145	8.23 sglSwapBuffers
146	8.24 sglSetDisplayStart
147	8.25 sglClear
148	8.26 sglHostToSurface
149	8.27 sglSurfaceToHost
150	8.28 sglCopyRect
151	8.29 sglBlit
152	8.30 sglScaleBlit
153	8.31 sglPushState
154	8.32 sglPopState
155	8.33 sglSetColor
156	8.34 sglSetBackgroundColor
157	8.35 sglSetRasterOp
158	8.36 sglSetWriteMask
159	8.37 sglSetLineStyle
160	8.38 sglSetLineWidth
161	8.39 sglSetFillPattern
162	8.40 sglSetPatternOrigin
163	8.41 sglLoadIdentity
164	8.42 sglLoadMatrix
165	8.43 sglMultiplyMatrix
166	8.44 sglTranslate
167	8.45 sglScale
168	8.46 sglRotate

Contents 5 of 6

Page	Topic
169	8.47 sglSetWindow
170	8.48 sglSetViewport
171	8.49 sglSetOrigin
172	8.50 sglSetClipRect
173	8.51 sglEnableClip
174	8.52 sglPoint
175	8.53 sglPoints
176	8.54 sglLine
177	8.55 sglPolyline
178	8.56 sglRectangle
179	8.57 sglFillRectangle
180	8.58 sglTriangle
181	8.59 sglFillTriangle
182	8.60 sglTriangles
183	8.61 sglPolygon
184	8.62 sglFillPolygon
185	8.63 sglCircle
186	8.64 sglArc
187	8.65 sglSetPaletteEntry
188	8.66 sglSetPaletteRange
189	8.67 sglGetPaletteEntry
190	8.68 sglSetColorMap
191	8.69 sglLoadFont
192	8.70 sglSelectFont
193	8.71 sglDestroyFont
194	8.72 sglSetTextColor
195	8.73 sglSetTextRotation
196	8.74 sglDrawText
197	8.75 sglMeasureText
198	8.76 sglLoadSymbolSet
199	8.77 sglDrawSymbol
200	8.78 sglCreateCursor
201	8.79 sglDestroyCursor
202	8.80 sglSetCursorImage
203	8.81 sglSetCursorPosition
204	8.82 sglShowCursor
205	8.83 sglWaitVBlank
206	8.84 sglGetStatus
207	8.85 sglGetLastError
208	8.86 sglSetErrorHandler
209	8.87 sglGetStatistics
210	8.88 sglBeginBatch
211	8.89 sglEndBatch
212	8.90 sglSetAcceleration
213	9.1 Acceleration Architecture
214	9.2 VDP-7 Coverage
215	9.3 VDP-6 Coverage
216	9.4 State Translation
217	9.5 Command Batching
218	9.6 Surface Placement
219	9.7 Format Constraints
220	9.8 Transform Constraints

Contents 6 of 6

Page	Topic
221	9.9 Clipping Strategy
222	9.10 Hardware-Only Qualification
223	9.11 Measuring Performance
224	9.12 Memory Bandwidth
225	9.13 Driver and Bridge Effects
226	9.14 Acceleration Checklist
227	10.1 Error Classes
228	10.2 Synchronous Errors
229	10.3 Asynchronous Errors
230	10.4 Device Lost Recovery
231	10.5 Surface Lost Recovery
232	10.6 Timeout Handling
233	10.7 Thread Safety
234	10.8 Error Handlers
235	10.9 Logging and Trace
236	10.10 Performance Measurement
237	10.11 Common Performance Traps
238	10.12 Field Diagnostic Checklist
239	11.1 Graphics Library 4.0 Compatibility
240	11.2 Changed Clipping Rules
241	11.3 Buffer Lifetime Changes
242	11.4 Revised Transform Validation
243	11.5 VDP-7 New Paths
244	11.6 VDP-6 Behavior
245	11.7 Vector Compatibility Library
246	11.8 Deprecated Aliases
247	11.9 Migration Test Plan
248	11.10 Deployment Migration Checklist
249	12.1 Basic Windowed Drawing
250	12.2 Strip Chart Trace
251	12.3 Double-Buffered Frame
252	12.4 Off-Screen Layer Composition
253	12.5 Indexed Palette Ramp
254	12.6 Technical Labels
255	12.7 Symbol Overlay
256	12.8 Host Image Upload
257	12.9 Hardware-Only Qualification
258	12.10 Recovery Skeleton
259	Appendix A - Public Types and Constants
260	Appendix B - Pixel Format Summary
261	Appendix C - Error Code Quick Reference
262	Appendix D - Capability Selectors
263	Appendix E - Raster Operation Summary
264	Appendix F - Rectangle and Clipping Rules
265	Appendix G - Function Index A-M
266	Appendix H - Function Index N-Z
267	Appendix I - Glossary
268	Appendix J - Support and Document Identification

Chapter 1 - Introduction

1.1 Library Overview

Graphics Library 4.1 is a stateful C graphics interface designed for technical visualization, instrumentation, engineering analysis and OEM display applications. It separates application drawing logic from device-register programming while preserving deterministic control over surfaces, transforms, clipping, synchronization and device capabilities.

Key Points

- Library Overview is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Library Overview, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.2 Software Stack and Components

The library is divided into the public application interface, host-side state and validation services, device-driver integration, font/symbol services and VDP-oriented acceleration backends. On Windows NT 4.0 the application never maps VDP registers directly.

Key Points

- Software Stack and Components is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Software Stack and Components, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.3 Supported Devices and Platforms

The primary supported platform is Windows NT 4.0 on qualified Sinneslöschen visualization systems using VDP-6 or VDP-7 display hardware. VDP-7 exposes the broadest acceleration coverage while VDP-6 remains supported for established OEM configurations.

Key Points

- Supported Devices and Platforms is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Supported Devices and Platforms, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.4 Execution Model

Most drawing calls are logically ordered but may be deferred. Graphics Library can batch state and graphics commands so that the host does not wait for each primitive to complete before submitting the next operation.

Key Points

- Execution Model is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Execution Model, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.5 Library State

Graphics state includes draw/read surfaces, logical colors, raster operation, write mask, line/fill attributes, transforms, viewport, clip rectangle, font/symbol selection, acceleration policy and synchronization options.

Key Points

- Library State is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Library State, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.6 Coordinate and Pixel Independence

Applications normally express geometry in user coordinates and colors as logical values. The library transforms geometry and converts colors according to the active viewport, surface format and device capabilities.

Key Points

- Coordinate and Pixel Independence is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Coordinate and Pixel Independence, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.7 Error Model

Most functions return `SGL_OK` or a negative `SGL_ERR_*` result. The library also records a per-thread or per-context last-error record and can invoke an optional error callback for asynchronous failures.

Key Points

- Error Model is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Error Model, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

1.8 Minimal Program

A minimal program opens a device, creates and makes current a context, acquires/binds a display surface, sets drawing state, emits primitives, establishes the required synchronization boundary, and releases objects in reverse order.

Key Points

- Minimal Program is defined at the library API boundary and should not be inferred from VDP register behavior.
- Applications should query capabilities when behavior can vary by device or driver revision.
- Qualified results must be preserved even when the implementation chooses a software fallback.

Programming Notes

For Minimal Program, prefer explicit state and documented lifecycle calls over assumptions based on a particular workstation. This keeps OEM and laboratory software portable between VDP-6 and VDP-7 systems.

2.1 Installation Components

The standard Windows NT package installs the runtime DLL, import library, C headers, sample source, diagnostic utilities, font/symbol resources and release notes.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Installation Components from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.2 Directory Layout

A typical installation separates system runtime files from developer headers, libraries, samples and documentation under the Sinneslöschen development tree.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Directory Layout from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.3 Header Files

Applications include sgl41.h for the public API; optional headers provide published symbols and compatibility aliases.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Header Files from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.4 Import Libraries and DLLs

The runtime is supplied as a dynamic library; applications link against the matching import library and should verify runtime version during initialization.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Import Libraries and DLLs from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.5 Compiler Settings

The public ABI uses 32-bit C calling conventions and documented structure packing compatible with supported Windows NT compilers.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Compiler Settings from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.6 Building a Console Sample

Console programs are useful for enumeration, off-screen rendering and automated verification because they remove user-interface dependencies.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Building a Console Sample from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.7 Building a Win32 Application

GUI applications normally create their window first, open the graphics device, then create library resources sized for the intended client or full-screen technical display.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Building a Win32 Application from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.8 Runtime Version Checking

sglGetVersion reports the installed runtime independently from the header version used to build the executable.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Runtime Version Checking from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.9 Driver Requirements

Graphics Library 4.1 depends on a qualified Sinneslöschen display/VDP driver and may reduce acceleration if the driver reports an older feature level.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Driver Requirements from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.10 Environment and Configuration

Normal applications require no private environment variables; diagnostic builds may use documented trace or validation settings.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Environment and Configuration from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.11 Redistributable Runtime

OEM redistribution is controlled by the applicable software agreement and requires the runtime, driver components and resource files to remain a qualified set.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Redistributable Runtime from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.12 Debug and Release Builds

Debug libraries add parameter checks and trace services but are not qualified for production timing.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Debug and Release Builds from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.13 Installation Verification

The supplied verification utility exercises device discovery, representative formats, drawing, blit and vertical-blank synchronization.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Installation Verification from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

2.14 Deployment Checklist

A supportable deployment records application, runtime, driver, VDP hardware, display mode and required font/symbol resources.

Key Points

- Use files from one qualified release set unless a Technical Bulletin authorizes mixing versions.
- Preserve release notes and version data with the deployed system.
- Verify the actual production image on representative hardware.

Programming Notes

Diagnose Deployment Checklist from the library/driver boundary outward. Confirm device detection, runtime version, driver revision and permissions before changing application graphics logic.

Chapter 3 - Programming Model

3.1 Object Model

Opaque handles represent devices, contexts, surfaces, fonts, cursors and symbol sets. The application may compare handles only as documented and must not inspect their internal representation.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Object Model are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.2 Device Lifetime

A device handle represents an installed logical graphics device and owns device capabilities and display-managed surfaces.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Device Lifetime are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.3 Context Lifetime

A context contains drawing state and command-submission state associated with one device.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Context Lifetime are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.4 Surface Lifetime

A surface represents rectangular pixel storage with dimensions, format, pitch, placement and usage flags.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Surface Lifetime are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.5 Current Context Rules

The current context is associated with the calling thread and defines which state later graphics calls modify.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Current Context Rules are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.6 Draw and Read Surfaces

The draw surface receives rendering results while the read surface supplies source pixels for transfer and readback functions.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Draw and Read Surfaces are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.7 State Change Ordering

Documented state changes are ordered before later dependent drawing even when redundant values are coalesced.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around State Change Ordering are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.8 Deferred Execution

A drawing call can return before hardware completion; input ownership and synchronization obligations follow the function-specific contract.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Deferred Execution are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.9 Flush and Finish

Flush submits pending work without waiting; Finish submits and waits for all prior work in the current context to complete.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Flush and Finish are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.10 Begin and End Batch

Batching allows the library to minimize host/driver transitions across a declared sequence without changing rendering order.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Begin and End Batch are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.11 State Stack

PushState and PopState save and restore selected graphics state categories, not allocated objects or pending work.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around State Stack are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.12 Default State

New contexts begin with identity transform, full-surface viewport/clip, COPY raster operation, full write mask, solid one-pixel lines and AUTO acceleration.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Default State are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.13 Capability Queries

Capabilities describe current backend features and limits but do not promise hardware acceleration for every combination of state and format.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Capability Queries are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.14 Device Reset

A driver or hardware reset can invalidate cached backend state and device-memory surfaces; applications must follow the documented recovery lifecycle.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Device Reset are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.15 Window Resize and Mode Change

Visible surface dimensions and pitch can change after mode transitions; viewport and clip state therefore require revalidation.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Window Resize and Mode Change are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.16 Display Ownership

Full-screen technical display ownership is established through the supported platform integration and does not make unrelated applications automatically cooperate.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Display Ownership are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.17 Threading Summary

Different threads can use different contexts subject to driver capability; shared object lifetime requires application synchronization.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Threading Summary are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

3.18 Determinism and Qualification

Rendering results are deterministic within documented precision, but call duration and acceleration path vary with device, mode, placement and state.

Key Points

- Treat all handles as library-owned tokens.
- Destroy dependent objects in documented order.
- Do not confuse API return with hardware completion.

Programming Notes

Most field failures around Determinism and Qualification are lifecycle or ordering errors rather than primitive math errors. Capture object ownership, current context, bound surfaces and the last synchronization boundary together.

4.1 Device Enumeration

Enumeration returns logical Sinneslöschen graphics devices in a stable order for the current process, not a permanent physical numbering scheme.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Device Enumeration, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.2 Device Information

Device information includes model, revision, display memory, formats, dimensions and acceleration capability flags.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Device Information, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.3 Opening a Device

Opening establishes a compatible driver connection but does not change display timing or clear the framebuffer.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Opening a Device, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.4 Closing a Device

Closing requires application-created dependent contexts and surfaces to be released first.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Closing a Device, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.5 Creating a Context

Context creation accepts optional preferences for state stack, command buffering and validation policy.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Creating a Context, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.6 Context Destruction

Destroying a context releases its host/driver state but does not imply a caller-requested completion boundary unless one is required for safe teardown.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Context Destruction, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.7 Front Surface

The front surface is a display-managed borrowed surface representing current scanout storage.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Front Surface, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.8 Back Surface

A double-buffered configuration may expose a display-managed back surface whose identity can change across swaps.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Back Surface, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.9 Off-Screen Surfaces

Off-screen surfaces hold intermediate images, glyph caches, layers, icons and page-flip staging data.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Off-Screen Surfaces, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.10 Surface Descriptors

Descriptors report width, height, pixel format, row pitch, placement, alignment and usage/capability flags.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Surface Descriptors, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.11 Surface Binding

Binding selects the current draw target but does not automatically reset transforms or application clipping.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Surface Binding, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.12 Read Surface Selection

The read surface can differ from the draw target and is used by copy/readback services.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Read Surface Selection, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.13 Surface Locking

LockSurface exposes CPU access when placement permits and may synchronize with earlier device drawing.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Surface Locking, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.14 Lock Flags

Read, write, discard and no-wait flags allow the application to describe intended host access precisely.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Lock Flags, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.15 Display Start and Panning

Display-start services select a scanout origin subject to format, alignment and vertical-blank rules.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Display Start and Panning, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.16 Surface Format Compatibility

Selected copy and transfer functions convert formats, but every source/destination pair is not guaranteed to be accelerated.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Surface Format Compatibility, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.17 Surface Loss

Device-memory storage can be lost across reset or mode reinitialization and must be reconstructed from application-owned data.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Surface Loss, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

4.18 Surface Design Checklist

Surface placement should match access pattern, format and recreation strategy rather than simply request device memory for every object.

Key Points

- Always use the width, format and pitch reported by the library.
- Do not destroy a bound, locked or in-flight object.
- Reacquire display-owned surface data after a mode transition.

Programming Notes

When diagnosing Surface Design Checklist, test odd widths, non-default pitches, clipped transfers and mode changes. Intermittent corruption often reveals an incorrect row-pitch or synchronization assumption.

Chapter 5 - Coordinates, Transforms and Clipping

5.1 Coordinate Pipeline

Geometry passes through model transform, logical window mapping, viewport conversion and clipping before device pixels are generated.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Coordinate Pipeline behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.2 User Coordinates

User coordinates are signed 32-bit application values and can represent engineering units or pixel-like values according to current mapping.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When User Coordinates behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.3 Identity Mapping

The default mapping treats one user unit as one device pixel in the current viewport.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Identity Mapping behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.4 Matrix Representation

SGL_MATRIX represents a 3 x 3 homogeneous two-dimensional transform using documented fixed-point fields.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Matrix Representation behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.5 Load and Multiply Order

LoadMatrix replaces the transform; MultiplyMatrix composes it using the library post-multiplication convention.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Load and Multiply Order behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.6 Translation

Translate composes a user-space offset into the current model transform.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Translation behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.7 Scaling

Scale applies independent fixed-point X and Y factors and can mirror an axis when negative.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Scaling behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.8 Rotation

Rotate composes a counter-clockwise fixed-point angle and uses exact orthogonal representations where possible.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Rotation behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.9 Window Mapping

SetWindow defines logical bounds mapped into the current viewport, useful for plots and instrumentation.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Window Mapping behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.10 Viewport Mapping

SetViewport defines the device-space destination rectangle for transformed coordinates.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Viewport Mapping behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.11 Origin

SetOrigin applies a final integer device offset for efficient panel relocation.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Origin behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.12 Clip Rectangle

SetClipRect establishes the application clip, intersected with immutable surface bounds.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Clip Rectangle behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.13 Clip Enable

Application clipping can be disabled, but surface-bound protection is always active.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Clip Enable behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.14 Rectangle Edge Rules

Transfer and fill rectangles use left/top inclusive and right/bottom exclusive bounds.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Rectangle Edge Rules behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.15 Line Endpoint Rules

Line and polyline endpoint inclusion is defined to avoid double-processing connected segment joints.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Line Endpoint Rules behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.16 Subpixel and Integer Paths

Public integer coordinates can be combined with fixed-point matrices and hardware subpixel setup where supported.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Subpixel and Integer Paths behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.17 Overflow and Range Checking

Unsafe transformed values are rejected before device submission rather than wrapping into invalid coordinates.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Overflow and Range Checking behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.18 Mirrored Coordinates

Negative scaling mirrors geometry; text and symbols follow their own orientation and baseline rules.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Mirrored Coordinates behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.19 Multi-Panel Layout

Window, viewport, origin and clip changes can implement dense engineering dashboards without recreating surfaces.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Multi-Panel Layout behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

5.20 Coordinate Checklist

Qualification should include negative coordinates, large ranges, clipping at every edge, zero-area rectangles, mirrored transforms and repeated resize/mode changes.

Key Points

- Transform and clip state belongs to the current context.
- Surface boundaries remain authoritative after every transform.
- Use one consistent rectangle convention at application module boundaries.

Programming Notes

When Coordinate Checklist behaves differently after resize or surface rebinding, record matrix, window, viewport, origin, clip and destination dimensions together. Primitive coordinates alone rarely explain the fault.

6.1 Logical Color Model

Logical color values are converted to indexed or direct-color destination representation according to surface format and color-map policy.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Logical Color Model, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.2 Indexed Color

INDEX8 surfaces store palette indices and can map logical RGB values through the active color map.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Indexed Color, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.3 Direct Color Formats

RGB565 and XRGB8888 store packed direct color with documented component precision.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Direct Color Formats, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.4 Palette Management

Palette functions update lookup-table entries without changing framebuffer indices.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Palette Management, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.5 Palette Synchronization

Large visible palette changes should occur during vertical blank or under forced blank when supported.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Palette Synchronization, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.6 Color Maps

Color-map objects preserve semantic application colors across indexed and direct-color configurations.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Color Maps, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.7 Write Mask

The write mask limits which stored destination bits/components can be modified by supported operations.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Write Mask, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.8 Raster Operations

COPY, XOR and other logical raster operations combine source/current color with stored destination values.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Raster Operations, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.9 Font Resources

Fonts are explicit library resources providing glyph data and metrics.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Font Resources, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.10 Technical Font and Symbol Library

Technical Font & Symbol Library 3.1 supplies stable engineering characters and instrumentation symbols.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Technical Font and Symbol Library, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.11 Glyph Caching

Frequently used glyphs can be cached in device surfaces for accelerated text rendering.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Glyph Caching, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.12 Text Baseline and Metrics

Text layout uses baseline, ascent, descent, advance and bounding metrics rather than character-count estimates.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Text Baseline and Metrics, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.13 Text Rotation

Orthogonal text rotation is broadly supported; arbitrary orientation can use software or extended VDP paths.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Text Rotation, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.14 Text Clipping

Glyph pixels are clipped normally without altering string advance.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Text Clipping, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.15 Symbol Rendering

Symbol identifiers are interpreted within a loaded published symbol set and rendered using text/symbol color state.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Symbol Rendering, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

6.16 Color and Text Checklist

Qualification should cover indexed/direct color, palette updates, clipping, missing glyphs, rotated text, resource version mismatch and cache recreation.

Key Points

- Color conversion depends on destination format.
- Font and symbol resources have explicit lifetime/version requirements.
- Visible palette or cache changes can require synchronization.

Programming Notes

For Color and Text Checklist, use measured metrics and published resource identifiers instead of hard-coded assumptions. This preserves layout across display formats and future resource updates.

Chapter 7 - Primitives and Image Operations

7.1 Primitive Overview

Primitives include points, lines, rectangles, triangles, polygons, arcs/ellipses, text/symbols and image transfers.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Primitive Overview, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.2 Points

Point operations support sampled plots and diagnostic markers using current point size and raster state.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Points, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.3 Lines

Lines use current color, raster operation, write mask, width and style and are a primary VDP acceleration path.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Lines, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.4 Polylines

Polyline processing shares setup across connected segments and avoids duplicate internal endpoint processing.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Polylines, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.5 Rectangles

Rectangle outline/fill calls use the library edge convention and optimized axis-aligned paths.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Rectangles, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.6 Triangles

Triangles are the fundamental accelerated filled polygon element on VDP-7.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Triangles, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.7 Polygons

General simple polygons may be triangulated by the library while self-intersection is not a portable fill case.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Polygons, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.8 Circles and Ellipses

Curves are generated device-independently and can use VDP vector segments or software paths according to transform/state.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Circles and Ellipses, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.9 Arcs

Arc operations support gauge, dial and polar-plot drawing with fixed-point angles.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Arcs, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.10 Fill Patterns

Repeated fill patterns and pattern origin support hatching and technical differentiation.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Fill Patterns, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.11 Screen-to-Screen Copy

CopyRect is overlap-safe and optimized when source and destination formats match.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Screen-to-Screen Copy, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.12 General Blit

Blit combines explicit surfaces, rectangles, format conversion, clipping and raster operation.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For General Blit, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.13 Scaled Blit

ScaleBlit maps source to differently sized destination with hardware support for qualified combinations.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Scaled Blit, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.14 Host Image Transfer

Host image services transfer application memory to/from library surfaces using explicit host pitch and format.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Host Image Transfer, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.15 Clear and Fill

Clear and full-surface fill use destination-specific fast paths.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Clear and Fill, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

7.16 Primitive Qualification Checklist

Every primitive should be tested with degenerate geometry, clipping, transforms, alternate formats and forced fallback/hardware-only modes.

Key Points

- All drawing is clipped to allocated surface bounds.
- Acceleration depends on the complete format/state/transform combination.
- Degenerate geometry follows each function page rather than undocumented driver behavior.

Programming Notes

For Primitive Qualification Checklist, batch compatible operations and avoid unnecessary state changes. On fast VDP hardware, host state churn can cost more than the primitive itself.

Chapter 8 - Function Reference

8.1 sglGetVersion

Synopsis

```
SGL_RESULT sglGetVersion(SGL_VERSION *version)
```

Purpose

Returns installed Graphics Library version information.

Parameters

Parameter	Description
version	receives major/minor/maintenance/build

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Returns installed Graphics Library version information. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
SGL_VERSION v = { sizeof(v) };
sglGetVersion(&v);
```

See Also

sglQueryCapability, sglGetDeviceInfo

8.2 sglEnumerateDevices

Synopsis

```
SGL_RESULT sglEnumerateDevices(uint32_t index, SGL_DEVICE_INFO *info)
```

Purpose

Enumerates logical Sinneslöschen graphics devices.

Parameters

Parameter	Description
index	zero-based enumeration index
info	receives device summary

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Enumerates logical Sinneslöschen graphics devices. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglEnumerateDevices(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglOpenDevice, sglGetDeviceInfo

8.3 sglOpenDevice

Synopsis

```
SGL_RESULT sglOpenDevice(uint32_t selector, SGL_DEVICE *device)
```

Purpose

Opens a logical graphics device.

Parameters

Parameter	Description
selector	device selector or default
device	receives device handle

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Opens a logical graphics device. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglOpenDevice(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglCloseDevice

8.4 sgICloseDevice

Synopsis

```
SGL_RESULT sgICloseDevice(SGL_DEVICE device)
```

Purpose

Closes an open graphics device.

Parameters

Parameter	Description
device	open device handle

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Closes an open graphics device. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sgICloseDevice(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sgIOpenDevice

8.5 sglGetDeviceInfo

Synopsis

```
SGL_RESULT sglGetDeviceInfo(SGL_DEVICE device, SGL_DEVICE_INFO *info)
```

Purpose

Returns detailed device information.

Parameters

Parameter	Description
device	open device
info	receives model/revision/memory/features

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Returns detailed device information. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglGetDeviceInfo(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglQueryCapability

8.6 sgICreateContext

Synopsis

```
SGL_RESULT sgICreateContext(SGL_DEVICE device, const SGL_CONTEXT_DESC *desc, SGL_CONTEXT *context)
```

Purpose

Creates a graphics context.

Parameters

Parameter	Description
device	open device
desc	optional descriptor
context	receives context handle

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Creates a graphics context The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sgICreateContext(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sgIDestroyContext, sgIMakeCurrent

8.7 sglDestroyContext

Synopsis

```
SGL_RESULT sglDestroyContext(SGL_CONTEXT context)
```

Purpose

Destroys a non-current context.

Parameters

Parameter	Description
context	context handle

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Destroys a non-current context The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglDestroyContext(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglCreateContext

8.8 sglMakeCurrent

Synopsis

```
SGL_RESULT sglMakeCurrent(SGL_CONTEXT context)
```

Purpose

Associates a context with the calling thread.

Parameters

Parameter	Description
context	context handle or NULL

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Associates a context with the calling thread. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglMakeCurrent(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglGetCurrentContext

8.9 sglGetCurrentContext

Synopsis

```
SGL_CONTEXT sglGetCurrentContext(void)
```

Purpose

Returns the calling thread's current context.

Parameters

This function has no parameters.

Return Value

Returns the current context handle, or NULL when no context is current.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Returns the calling thread's current context. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglGetCurrentContext(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglMakeCurrent

8.10 sglFlush

Synopsis

```
SGL_RESULT sglFlush(void)
```

Purpose

Submits pending work without waiting.

Parameters

This function has no parameters.

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Submits pending work without waiting The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
sglFlush();
```

See Also

sglFinish

8.11 sglFinish

Synopsis

```
SGL_RESULT sglFinish(void)
```

Purpose

Submits pending work and waits for completion.

Parameters

This function has no parameters.

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Submits pending work and waits for completion. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
sglFinish();
```

See Also

sglFlush, sglWaitVBlank

8.12 sglResetContext

Synopsis

```
SGL_RESULT sglResetContext(uint32_t flags)
```

Purpose

Resets selected context state to defaults.

Parameters

Parameter	Description
flags	reset category mask

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Resets selected context state to defaults. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglResetContext(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglPushState, sglPopState

8.13 sglQueryCapability

Synopsis

```
SGL_RESULT sglQueryCapability(uint32_t capability, uint32_t *value)
```

Purpose

Queries a feature or implementation limit.

Parameters

Parameter	Description
capability	SGL_CAP selector
value	receives value

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Queries a feature or implementation limit The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
/* See parameter descriptions above. */
result = sglQueryCapability(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetAcceleration

8.14 sglGetFrontSurface

Synopsis

```
SGL_RESULT sglGetFrontSurface(SGL_DEVICE device, SGL_SURFACE *surface)
```

Purpose

Returns the display front surface.

Parameters

Parameter	Description
device	open device
surface	receives borrowed surface

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Returns the display front surface. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglGetFrontSurface(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglGetBackSurface

8.15 sglCreateSurface

Synopsis

```
SGL_RESULT sglCreateSurface(const SGL_SURFACE_DESC *desc, SGL_SURFACE *surface)
```

Purpose

Creates an application-owned surface.

Parameters

Parameter	Description
desc	dimensions/format/placement
surface	receives handle

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Creates an application-owned surface. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglCreateSurface(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglDestroySurface

8.16 sglDestroySurface

Synopsis

```
SGL_RESULT sglDestroySurface(SGL_SURFACE surface)
```

Purpose

Destroys an application-owned surface.

Parameters

Parameter	Description
surface	surface handle

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Destroys an application-owned surface. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglDestroySurface(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglCreateSurface

8.17 sglGetSurfaceInfo

Synopsis

```
SGL_RESULT sglGetSurfaceInfo(SGL_SURFACE surface, SGL_SURFACE_INFO *info)
```

Purpose

Returns dimensions/format/pitch/placement.

Parameters

Parameter	Description
surface	surface handle
info	receives descriptor

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Returns dimensions/format/pitch/placement The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglGetSurfaceInfo(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglLockSurface

8.18 sglBindSurface

Synopsis

```
SGL_RESULT sglBindSurface(SGL_SURFACE surface)
```

Purpose

Selects the current draw surface.

Parameters

Parameter	Description
surface	destination surface

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Selects the current draw surface. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
sglBindSurface(plotSurface);
```

See Also

sglSetReadSurface

8.19 sglSetReadSurface

Synopsis

```
SGL_RESULT sglSetReadSurface(SGL_SURFACE surface)
```

Purpose

Selects the current read/source surface.

Parameters

Parameter	Description
surface	source surface

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Selects the current read/source surface. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
sglSetReadSurface(value);
```

See Also

sglBindSurface

8.20 sglLockSurface

Synopsis

```
SGL_RESULT sglLockSurface(SGL_SURFACE surface, uint32_t flags, SGL_LOCKED_SURFACE *lock)
```

Purpose

Maps a surface for CPU access.

Parameters

Parameter	Description
surface	surface handle
flags	read/write/discard/no-wait
lock	receives pointer/pitch

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Maps a surface for CPU access. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglLockSurface(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglUnlockSurface

8.21 sglUnlockSurface

Synopsis

```
SGL_RESULT sglUnlockSurface(SGL_SURFACE surface)
```

Purpose

Releases a CPU surface lock.

Parameters

Parameter	Description
surface	locked surface

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Releases a CPU surface lock. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglUnlockSurface(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglLockSurface

8.22 sglSetDrawBuffer

Synopsis

```
SGL_RESULT sglSetDrawBuffer(uint32_t buffer)
```

Purpose

Selects front/back display buffer as draw target.

Parameters

Parameter	Description
buffer	front or back

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Selects front/back display buffer as draw target. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
sglSetDrawBuffer(value);
```

See Also

sglSwapBuffers

8.23 sglSwapBuffers

Synopsis

```
SGL_RESULT sglSwapBuffers(uint32_t flags)
```

Purpose

Presents the back buffer.

Parameters

Parameter	Description
flags	presentation/synchronization flags

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Presents the back buffer The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglSwapBuffers(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglWaitVBlank

8.24 sglSetDisplayStart

Synopsis

```
SGL_RESULT sglSetDisplayStart(SGL_SURFACE surface, int32_t x, int32_t y, uint32_t flags)
```

Purpose

Selects display scanout origin.

Parameters

Parameter	Description
surface	display-capable surface
x	origin x
y	origin y
flags	sync options

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Selects display scanout origin The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglSetDisplayStart(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSwapBuffers

8.25 sglClear

Synopsis

```
SGL_RESULT sglClear(uint32_t flags)
```

Purpose

Clears current draw target.

Parameters

Parameter	Description
flags	clear target mask

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Clears current draw target The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglClear(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetBackgroundColor

8.26 sglHostToSurface

Synopsis

```
SGL_RESULT sglHostToSurface(const SGL_HOST_IMAGE *src, SGL_SURFACE dst, const SGL_RECT *dstRect)
```

Purpose

Transfers host image into a surface.

Parameters

Parameter	Description
src	host image descriptor
dst	destination
dstRect	destination rectangle

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Transfers host image into a surface. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglHostToSurface(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSurfaceToHost

8.27 sglSurfaceToHost

Synopsis

```
SGL_RESULT sglSurfaceToHost(SGL_SURFACE src, const SGL_RECT *srcRect, SGL_HOST_IMAGE *dst)
```

Purpose

Reads surface pixels into host memory.

Parameters

Parameter	Description
src	source surface
srcRect	source rectangle
dst	host image destination

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Reads surface pixels into host memory. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglSurfaceToHost(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglHostToSurface

8.28 sglCopyRect

Synopsis

```
SGL_RESULT sglCopyRect(const SGL_RECT *srcRect, const SGL_POINT *dstPoint)
```

Purpose

Copies between current read/draw surfaces.

Parameters

Parameter	Description
srcRect	source rectangle
dstPoint	destination origin

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Copies between current read/draw surfaces. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglCopyRect(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglBlit

8.29 sglBlit

Synopsis

```
SGL_RESULT sglBlit(const SGL_BLIT_DESC *blit)
```

Purpose

Performs generalized image transfer.

Parameters

Parameter	Description
blit	source/destination/operation descriptor

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Performs generalized image transfer. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglBlit(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglCopyRect, sglScaleBlit

8.30 sglScaleBlit

Synopsis

```
SGL_RESULT sglScaleBlit(const SGL_BLIT_DESC *blit, uint32_t filter)
```

Purpose

Copies and scales an image.

Parameters

Parameter	Description
blit	transfer descriptor
filter	scaling filter

Return Value

SGL_OK on success.

Detailed Behavior

Surface and transfer behavior is defined in terms of explicit dimensions, row pitch, placement and pixel format. The library preserves surface bounds and format conversion rules even when it selects a different backend path. Copies and scales an image The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include invalid/lost surface, incompatible format, range/alignment violation, busy/locked object or missing display capability.

Performance / Synchronization

Device-local paths are fastest when formats match; CPU locking/readback and format conversion can serialize or stage transfers.

Example

```
/* See parameter descriptions above. */
result = sglScaleBlit(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglBlit

8.31 sglPushState

Synopsis

```
SGL_RESULT sglPushState(uint32_t mask)
```

Purpose

Pushes selected graphics state.

Parameters

Parameter	Description
mask	state category mask

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Pushes selected graphics state. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglPushState(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglPopState

8.32 sglPopState

Synopsis

```
SGL_RESULT sglPopState(void)
```

Purpose

Restores most recently pushed state.

Parameters

This function has no parameters.

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Restores most recently pushed state. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglPopState(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglPushState

8.33 sglSetColor

Synopsis

```
SGL_RESULT sglSetColor(SGL_COLOR color)
```

Purpose

Sets current foreground color.

Parameters

Parameter	Description
color	logical color

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Sets current foreground color The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
sglSetColor(value);
```

See Also

sglSetBackgroundColor

8.34 sglSetBackgroundColor

Synopsis

```
SGL_RESULT sglSetBackgroundColor(SGL_COLOR color)
```

Purpose

Sets background/clear color.

Parameters

Parameter	Description
color	logical color

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Sets background/clear color The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
sglSetBackgroundColor(value);
```

See Also

sglClear

8.35 sglSetRasterOp

Synopsis

```
SGL_RESULT sglSetRasterOp(uint32_t rop)
```

Purpose

Sets logical raster operation.

Parameters

Parameter	Description
rop	SGL_ROP selector

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Sets logical raster operation. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
sglSetRasterOp(value);
```

See Also

sglSetWriteMask

8.36 sglSetWriteMask

Synopsis

```
SGL_RESULT sglSetWriteMask(uint32_t mask)
```

Purpose

Sets destination write mask.

Parameters

Parameter	Description
mask	format-dependent write mask

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Sets destination write mask The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
sglSetWriteMask(value);
```

See Also

sglSetRasterOp

8.37 sglSetLineStyle

Synopsis

```
SGL_RESULT sglSetLineStyle(uint32_t pattern, uint32_t repeat)
```

Purpose

Sets line stipple pattern.

Parameters

Parameter	Description
pattern	16-bit pattern
repeat	repeat factor

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Sets line stipple pattern The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
sglSetLineStyle(value, value2);
```

See Also

sglSetLineWidth

8.38 sglSetLineWidth

Synopsis

```
SGL_RESULT sglSetLineWidth(uint32_t width)
```

Purpose

Sets line width.

Parameters

Parameter	Description
width	device-pixel line width

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Sets line width The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
sglSetLineWidth(value);
```

See Also

sglSetLineStyle

8.39 sglSetFillPattern

Synopsis

```
SGL_RESULT sglSetFillPattern(const SGL_PATTERN *pattern)
```

Purpose

Selects fill pattern.

Parameters

Parameter	Description
pattern	pattern or NULL for solid

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Selects fill pattern The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
sglSetFillPattern(value);
```

See Also

sglSetPatternOrigin

8.40 sglSetPatternOrigin

Synopsis

```
SGL_RESULT sglSetPatternOrigin(int32_t x, int32_t y)
```

Purpose

Sets fill-pattern origin.

Parameters

Parameter	Description
x	origin x
y	origin y

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Sets fill-pattern origin The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
sglSetPatternOrigin(value, value2);
```

See Also

sglSetFillPattern

8.41 sglLoadIdentity

Synopsis

```
SGL_RESULT sglLoadIdentity(void)
```

Purpose

Loads identity model transform.

Parameters

This function has no parameters.

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Loads identity model transform The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
sglLoadIdentity();
```

See Also

sglLoadMatrix

8.42 sglLoadMatrix

Synopsis

```
SGL_RESULT sglLoadMatrix(const SGL_MATRIX *matrix)
```

Purpose

Loads model transform.

Parameters

Parameter	Description
matrix	matrix pointer

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Loads model transform The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */
result = sglLoadMatrix(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglMultiplyMatrix

8.43 sglMultiplyMatrix

Synopsis

```
SGL_RESULT sglMultiplyMatrix(const SGL_MATRIX *matrix)
```

Purpose

Composes matrix with current transform.

Parameters

Parameter	Description
matrix	matrix pointer

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Composes matrix with current transform. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */  
result = sglMultiplyMatrix(...);  
if (result != SGL_OK) handle_error(result);
```

See Also

sglLoadMatrix

8.44 sglTranslate

Synopsis

```
SGL_RESULT sglTranslate(int32_t x, int32_t y)
```

Purpose

Composes translation.

Parameters

Parameter	Description
x	x offset
y	y offset

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Composes translation The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */
result = sglTranslate(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetOrigin

8.45 sglScale

Synopsis

```
SGL_RESULT sglScale(int32_t xScale, int32_t yScale)
```

Purpose

Composes fixed-point scale.

Parameters

Parameter	Description
xScale	x scale
yScale	y scale

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Composes fixed-point scale. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */
result = sglScale(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglRotate

8.46 sglRotate

Synopsis

```
SGL_RESULT sglRotate(int32_t angle)
```

Purpose

Composes fixed-point rotation.

Parameters

Parameter	Description
angle	16.16 degrees

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Composes fixed-point rotation The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */
result = sglRotate(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglScale

8.47 sglSetWindow

Synopsis

```
SGL_RESULT sglSetWindow(int32_t left, int32_t top, int32_t right, int32_t bottom)
```

Purpose

Defines logical window mapping.

Parameters

Parameter	Description
left	left
top	top
right	right
bottom	bottom

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Defines logical window mapping. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */
result = sglSetWindow(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetViewport

8.48 sglSetViewport

Synopsis

```
SGL_RESULT sglSetViewport(const SGL_RECT *rect)
```

Purpose

Sets device-space viewport.

Parameters

Parameter	Description
rect	viewport rectangle

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Sets device-space viewport The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
sglSetViewport(value);
```

See Also

sglSetWindow

8.49 sglSetOrigin

Synopsis

```
SGL_RESULT sglSetOrigin(int32_t x, int32_t y)
```

Purpose

Sets device-space origin offset.

Parameters

Parameter	Description
x	x offset
y	y offset

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Sets device-space origin offset The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
sglSetOrigin(value, value2);
```

See Also

sglTranslate

8.50 sglSetClipRect

Synopsis

```
SGL_RESULT sglSetClipRect(const SGL_RECT *rect)
```

Purpose

Sets application clip rectangle.

Parameters

Parameter	Description
rect	clip bounds

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Sets application clip rectangle The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
sglSetClipRect(value);
```

See Also

sglEnableClip

8.51 sglEnableClip

Synopsis

```
SGL_RESULT sglEnableClip(int enable)
```

Purpose

Enables/disables application clipping.

Parameters

Parameter	Description
enable	nonzero enables

Return Value

SGL_OK on success.

Detailed Behavior

Transform and clipping state belongs to the current context and is applied in documented order before rasterization. Surface boundary protection remains active regardless of application clip state. Enables/disables application clipping The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are invalid pointers/enums, zero or reversed invalid bounds, representational overflow and missing current context/surface.

Performance / Synchronization

Transform state is usually deferred/coalesced. Repeatedly rebuilding equivalent matrices or changing clip for every small primitive adds host overhead.

Example

```
/* See parameter descriptions above. */
result = sglEnableClip(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetClipRect

8.52 sglPoint

Synopsis

```
SGL_RESULT sglPoint(int32_t x, int32_t y)
```

Purpose

Draws one point.

Parameters

Parameter	Description
x	user x
y	user y

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws one point The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglPoint(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglPoints

8.53 sglPoints

Synopsis

```
SGL_RESULT sglPoints(const SGL_POINT *points, uint32_t count)
```

Purpose

Draws an array of points.

Parameters

Parameter	Description
points	point array
count	number of points

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws an array of points. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglPoints(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglPoint

8.54 sglLine

Synopsis

```
SGL_RESULT sglLine(int32_t x0, int32_t y0, int32_t x1, int32_t y1)
```

Purpose

Draws one line segment.

Parameters

Parameter	Description
x0	start x
y0	start y
x1	end x
y1	end y

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws one line segment The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
sglLine(20, 20, 400, 300);
```

See Also

sglPolyline

8.55 sglPolyline

Synopsis

```
SGL_RESULT sglPolyline(const SGL_POINT *points, uint32_t count)
```

Purpose

Draws connected line segments.

Parameters

Parameter	Description
points	vertex array
count	vertex count

Return Value

SGL_OK on success.

Detailed Behavior

This operation participates in the Graphics Library object/context lifecycle. Handles are opaque and validated against the current process and owning device. The call does not expose or require VDP register programming. Draws connected line segments The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Invalid handle, pointer, structure size, object-state or device-compatibility conditions are reported before unsafe driver submission.

Performance / Synchronization

Object-management calls are not intended for per-primitive use; create/reuse long-lived objects where practical.

Example

```
/* See parameter descriptions above. */
result = sglPolyline(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglLine

8.56 sglRectangle

Synopsis

```
SGL_RESULT sglRectangle(const SGL_RECT *rect)
```

Purpose

Draws rectangle outline.

Parameters

Parameter	Description
rect	rectangle bounds

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws rectangle outline The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglRectangle(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglFillRectangle

8.57 sglFillRectangle

Synopsis

```
SGL_RESULT sglFillRectangle(const SGL_RECT *rect)
```

Purpose

Fills rectangle.

Parameters

Parameter	Description
rect	rectangle bounds

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Fills rectangle The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglFillRectangle(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglRectangle

8.58 sglTriangle

Synopsis

```
SGL_RESULT sglTriangle(const SGL_POINT *a, const SGL_POINT *b, const SGL_POINT *c)
```

Purpose

Draws triangle outline.

Parameters

Parameter	Description
a	first vertex
b	second vertex
c	third vertex

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws triangle outline The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */  
result = sglTriangle(...);  
if (result != SGL_OK) handle_error(result);
```

See Also

sglFillTriangle

8.59 sglFillTriangle

Synopsis

```
SGL_RESULT sglFillTriangle(const SGL_POINT *a, const SGL_POINT *b, const SGL_POINT *c)
```

Purpose

Fills triangle.

Parameters

Parameter	Description
a	first vertex
b	second vertex
c	third vertex

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Fills triangle The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */  
result = sglFillTriangle(...);  
if (result != SGL_OK) handle_error(result);
```

See Also

sglTriangles

8.60 sglTriangles

Synopsis

```
SGL_RESULT sglTriangles(const SGL_POINT *points, uint32_t triangleCount)
```

Purpose

Draws/fills independent triangle list.

Parameters

Parameter	Description
points	three vertices per triangle
triangleCount	number of triangles

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws/fills independent triangle list. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglTriangles(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglFillTriangle

8.61 sglPolygon

Synopsis

```
SGL_RESULT sglPolygon(const SGL_POINT *points, uint32_t count)
```

Purpose

Draws closed polygon outline.

Parameters

Parameter	Description
points	vertex array
count	vertex count

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws closed polygon outline. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglPolygon(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglFillPolygon

8.62 sglFillPolygon

Synopsis

```
SGL_RESULT sglFillPolygon(const SGL_POINT *points, uint32_t count)
```

Purpose

Fills polygon.

Parameters

Parameter	Description
points	vertex array
count	vertex count

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Fills polygon The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */  
result = sglFillPolygon(...);  
if (result != SGL_OK) handle_error(result);
```

See Also

sglTriangles

8.63 sglCircle

Synopsis

```
SGL_RESULT sglCircle(int32_t cx, int32_t cy, int32_t radius)
```

Purpose

Draws circle outline.

Parameters

Parameter	Description
cx	center x
cy	center y
radius	radius

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws circle outline The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglCircle(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglEllipse

8.64 sglArc

Synopsis

```
SGL_RESULT sglArc(int32_t cx, int32_t cy, int32_t rx, int32_t ry, int32_t startAngle, int32_t sweepAngle)
```

Purpose

Draws elliptical arc.

Parameters

Parameter	Description
cx	center x
cy	center y
rx	x radius
ry	y radius
startAngle	start angle
sweepAngle	sweep angle

Return Value

SGL_OK on success.

Detailed Behavior

The primitive uses current color, raster operation, write mask, transform, viewport, origin, clip and applicable line/fill state. Geometry is validated and clipped before pixels are committed. Draws elliptical arc The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures are no current context/surface, invalid pointer/count or transformed coordinate range. Complex fill can also report unsupported geometry.

Performance / Synchronization

Batch arrays/connected geometry where possible. One-pixel lines, rectangles and triangles have the strongest VDP acceleration coverage.

Example

```
/* See parameter descriptions above. */
result = sglArc(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglEllipse

8.65 sglSetPaletteEntry

Synopsis

```
SGL_RESULT sglSetPaletteEntry(uint32_t index, uint8_t r, uint8_t g, uint8_t b)
```

Purpose

Sets one palette entry.

Parameters

Parameter	Description
index	palette index
r	red
g	green
b	blue

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Sets one palette entry The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglSetPaletteEntry(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetPaletteRange

8.66 sglSetPaletteRange

Synopsis

```
SGL_RESULT sglSetPaletteRange(uint32_t first, uint32_t count, const SGL_RGB8 *entries)
```

Purpose

Updates a palette range.

Parameters

Parameter	Description
first	first index
count	entry count
entries	RGB array

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Updates a palette range The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglSetPaletteRange(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetPaletteEntry

8.67 sglGetPaletteEntry

Synopsis

```
SGL_RESULT sglGetPaletteEntry(uint32_t index, SGL_RGB8 *entry)
```

Purpose

Returns one palette entry.

Parameters

Parameter	Description
index	palette index
entry	receives RGB

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Returns one palette entry. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglGetPaletteEntry(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetPaletteEntry

8.68 sglSetColorMap

Synopsis

```
SGL_RESULT sglSetColorMap(const SGL_COLORMAP *map)
```

Purpose

Selects logical-to-device color mapping.

Parameters

Parameter	Description
map	color map or NULL

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Selects logical-to-device color mapping. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
sglSetColorMap(value);
```

See Also

sglSetColor

8.69 sglLoadFont

Synopsis

```
SGL_RESULT sglLoadFont(const char *name, uint32_t size, uint32_t flags, SGL_FONT *font)
```

Purpose

Loads a supported font resource.

Parameters

Parameter	Description
name	font name
size	nominal size
flags	style flags
font	receives handle

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Loads a supported font resource. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglLoadFont(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSelectFont

8.70 sglSelectFont

Synopsis

```
SGL_RESULT sglSelectFont(SGL_FONT font)
```

Purpose

Selects current font.

Parameters

Parameter	Description
font	font handle

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Selects current font The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglSelectFont(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglLoadFont

8.71 sglDestroyFont

Synopsis

```
SGL_RESULT sglDestroyFont(SGL_FONT font)
```

Purpose

Releases a font resource.

Parameters

Parameter	Description
font	font handle

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Releases a font resource The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglDestroyFont(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglLoadFont

8.72 sglSetTextColor

Synopsis

```
SGL_RESULT sglSetTextColor(SGL_COLOR color)
```

Purpose

Sets text/symbol color.

Parameters

Parameter	Description
color	logical color

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Sets text/symbol color The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
sglSetTextColor(value);
```

See Also

sglDrawText

8.73 sglSetTextRotation

Synopsis

```
SGL_RESULT sglSetTextRotation(int32_t angle)
```

Purpose

Sets text orientation.

Parameters

Parameter	Description
angle	16.16 degrees

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Sets text orientation The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
sglSetTextRotation(value);
```

See Also

sglDrawText

8.74 sgldrawText

Synopsis

```
SGL_RESULT sgldrawText(int32_t x, int32_t y, const char *text)
```

Purpose

Draws a text string.

Parameters

Parameter	Description
x	baseline x
y	baseline y
text	string

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Draws a text string The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
sgldrawText(40, 60, "FLOW RATE");
```

See Also

sglMeasureText

8.75 sglMeasureText

Synopsis

```
SGL_RESULT sglMeasureText(const char *text, SGL_TEXT_METRICS *metrics)
```

Purpose

Measures text without drawing.

Parameters

Parameter	Description
text	string
metrics	receives metrics

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Measures text without drawing. The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglMeasureText(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglDrawText

8.76 sglLoadSymbolSet

Synopsis

```
SGL_RESULT sglLoadSymbolSet(const char *name, SGL_SYMBOL_SET *set)
```

Purpose

Loads a technical symbol set.

Parameters

Parameter	Description
name	set resource name
set	receives handle

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Loads a technical symbol set The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglLoadSymbolSet(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglDrawSymbol

8.77 sglDrawSymbol

Synopsis

```
SGL_RESULT sglDrawSymbol(SGL_SYMBOL_SET set, uint32_t symbol, int32_t x, int32_t y, uint32_t size)
```

Purpose

Draws a technical symbol.

Parameters

Parameter	Description
set	symbol set
symbol	identifier
x	origin x
y	origin y
size	nominal size

Return Value

SGL_OK on success.

Detailed Behavior

Color, font and symbol services are device-independent at the API level and are converted/cached according to destination format and selected resources. Published resource identifiers and metrics remain the application contract. Draws a technical symbol The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid resource handle, missing font/symbol, unsupported format, encoding error or indexed-mode requirement.

Performance / Synchronization

Resource loading and palette-wide updates should be amortized. Reuse fonts, symbol sets and palette ranges rather than rebuilding them each frame.

Example

```
/* See parameter descriptions above. */
result = sglDrawSymbol(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglLoadSymbolSet

8.78 sglCreateCursor

Synopsis

```
SGL_RESULT sglCreateCursor(const SGL_CURSOR_DESC *desc, SGL_CURSOR *cursor)
```

Purpose

Creates cursor object.

Parameters

Parameter	Description
desc	cursor descriptor
cursor	receives handle

Return Value

SGL_OK on success.

Detailed Behavior

Cursor coordinates and image resources are display-oriented rather than transformed user geometry. Qualified VDP hardware cursors are used when size/format permit; software paths can be selected otherwise. Creates cursor object The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid cursor handle/image, unsupported format, device mismatch or coordinate/range limitation.

Performance / Synchronization

Move an existing cursor instead of recreating it. Hardware position updates are designed for frequent use.

Example

```
/* See parameter descriptions above. */
result = sglCreateCursor(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglDestroyCursor

8.79 sglDestroyCursor

Synopsis

```
SGL_RESULT sglDestroyCursor(SGL_CURSOR cursor)
```

Purpose

Destroys inactive cursor.

Parameters

Parameter	Description
cursor	cursor handle

Return Value

SGL_OK on success.

Detailed Behavior

Cursor coordinates and image resources are display-oriented rather than transformed user geometry. Qualified VDP hardware cursors are used when size/format permit; software paths can be selected otherwise. Destroys inactive cursor The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid cursor handle/image, unsupported format, device mismatch or coordinate/range limitation.

Performance / Synchronization

Move an existing cursor instead of recreating it. Hardware position updates are designed for frequent use.

Example

```
/* See parameter descriptions above. */
result = sglDestroyCursor(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglCreateCursor

8.80 sglSetCursorImage

Synopsis

```
SGL_RESULT sglSetCursorImage(SGL_CURSOR cursor, const SGL_CURSOR_IMAGE *image)
```

Purpose

Replaces cursor image.

Parameters

Parameter	Description
cursor	cursor handle
image	new image descriptor

Return Value

SGL_OK on success.

Detailed Behavior

Cursor coordinates and image resources are display-oriented rather than transformed user geometry. Qualified VDP hardware cursors are used when size/format permit; software paths can be selected otherwise. Replaces cursor image The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid cursor handle/image, unsupported format, device mismatch or coordinate/range limitation.

Performance / Synchronization

Move an existing cursor instead of recreating it. Hardware position updates are designed for frequent use.

Example

```
sglSetCursorImage(value, value2);
```

See Also

sglSetCursorPosition

8.81 sglSetCursorPosition

Synopsis

```
SGL_RESULT sglSetCursorPosition(SGL_CURSOR cursor, int32_t x, int32_t y)
```

Purpose

Moves cursor.

Parameters

Parameter	Description
cursor	cursor handle
x	display x
y	display y

Return Value

SGL_OK on success.

Detailed Behavior

Cursor coordinates and image resources are display-oriented rather than transformed user geometry. Qualified VDP hardware cursors are used when size/format permit; software paths can be selected otherwise. Moves cursor The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid cursor handle/image, unsupported format, device mismatch or coordinate/range limitation.

Performance / Synchronization

Move an existing cursor instead of recreating it. Hardware position updates are designed for frequent use.

Example

```
/* See parameter descriptions above. */
result = sglSetCursorPosition(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglShowCursor

8.82 sglShowCursor

Synopsis

```
SGL_RESULT sglShowCursor(SGL_CURSOR cursor, int visible)
```

Purpose

Shows/hides cursor.

Parameters

Parameter	Description
cursor	cursor handle
visible	nonzero shows

Return Value

SGL_OK on success.

Detailed Behavior

Cursor coordinates and image resources are display-oriented rather than transformed user geometry. Qualified VDP hardware cursors are used when size/format permit; software paths can be selected otherwise. Shows/hides cursor The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical failures include invalid cursor handle/image, unsupported format, device mismatch or coordinate/range limitation.

Performance / Synchronization

Move an existing cursor instead of recreating it. Hardware position updates are designed for frequent use.

Example

```
/* See parameter descriptions above. */
result = sglShowCursor(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetCursorPosition

8.83 sglWaitVBlank

Synopsis

```
SGL_RESULT sglWaitVBlank(uint32_t flags, uint32_t timeoutMs)
```

Purpose

Waits for vertical-blank phase.

Parameters

Parameter	Description
flags	phase selector
timeoutMs	maximum wait

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Waits for vertical-blank phase The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
/* See parameter descriptions above. */
result = sglWaitVBlank(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSwapBuffers, sglFinish

8.84 sglGetStatus

Synopsis

```
SGL_RESULT sglGetStatus(SGL_STATUS *status)
```

Purpose

Returns logical library/device status.

Parameters

Parameter	Description
status	receives status

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Returns logical library/device status The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
/* See parameter descriptions above. */  
result = sglGetStatus(...);  
if (result != SGL_OK) handle_error(result);
```

See Also

sglGetStatistics

8.85 sglGetLastError

Synopsis

```
SGL_RESULT sglGetLastError(SGL_ERROR_INFO *info)
```

Purpose

Returns latest error record.

Parameters

Parameter	Description
info	receives error info

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Returns latest error record The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
/* See parameter descriptions above. */
result = sglGetLastError(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglSetErrorHandler

8.86 sglSetErrorHandler

Synopsis

```
SGL_RESULT sglSetErrorHandler(SGL_ERROR_HANDLER handler, void *user)
```

Purpose

Installs optional error callback.

Parameters

Parameter	Description
handler	callback or NULL
user	caller value

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Installs optional error callback The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
sglSetErrorHandler(value, value2);
```

See Also

sglGetLastError

8.87 sglGetStatistics

Synopsis

```
SGL_RESULT sglGetStatistics(SGL_STATISTICS *stats)
```

Purpose

Returns diagnostic counters.

Parameters

Parameter	Description
stats	receives statistics

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Returns diagnostic counters The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
/* See parameter descriptions above. */
result = sglGetStatistics(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglResetStatistics

8.88 sglBeginBatch

Synopsis

```
SGL_RESULT sglBeginBatch(uint32_t flags)
```

Purpose

Begins command batching scope.

Parameters

Parameter	Description
flags	batch options

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Begins command batching scope The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
/* See parameter descriptions above. */
result = sglBeginBatch(...);
if (result != SGL_OK) handle_error(result);
```

See Also

sglEndBatch

8.89 sglEndBatch

Synopsis

```
SGL_RESULT sglEndBatch(void)
```

Purpose

Ends command batching scope.

Parameters

This function has no parameters.

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Ends command batching scope The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
sglEndBatch();
```

See Also

sglBeginBatch

8.90 sglSetAcceleration

Synopsis

```
SGL_RESULT sglSetAcceleration(uint32_t policy)
```

Purpose

Selects acceleration policy.

Parameters

Parameter	Description
policy	SGL_ACCEL selector

Return Value

SGL_OK on success.

Detailed Behavior

Synchronization and diagnostic services define ordering/completion or observe library/backend state. They do not change drawing results except through explicit acceleration or error-policy selection. Selects acceleration policy The operation is ordered relative to prior and subsequent calls according to the current context rules. Inputs are validated before the implementation emits device work that could violate surface or hardware limits.

Error Conditions

Typical errors include no current context, timeout, device lost, invalid selector/structure or asynchronous backend failure surfaced at a boundary.

Performance / Synchronization

Use the weakest synchronization that satisfies the requirement. Excessive Finish/wait operations can eliminate deferred execution benefits.

Example

```
sglSetAcceleration(value);
```

See Also

sglQueryCapability

9.1 Acceleration Architecture

The library translates eligible operations into VDP command/state sequences while retaining a software path for combinations not implemented by installed hardware.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Acceleration Architecture, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.2 VDP-7 Coverage

VDP-7 accelerates the broadest Graphics Library 4.1 feature set, including line, polygon, clipping, blit, scaling, palette/cursor and display synchronization operations.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For VDP-7 Coverage, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.3 VDP-6 Coverage

VDP-6 accelerates core line, rectangle, blit and raster functions but lacks or restricts several VDP-7 operations.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For VDP-6 Coverage, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.4 State Translation

Context state is tracked lazily and only required device state is emitted before dependent accelerated work.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For State Translation, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.5 Command Batching

VDP command packets are accumulated and submitted when capacity, Flush, EndBatch or synchronization requires it.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Command Batching, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.6 Surface Placement

Device-memory surfaces favor rendering throughput while host surfaces favor frequent CPU generation/access.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Surface Placement, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.7 Format Constraints

Acceleration depends on source/destination format, raster operation, write mask and conversion requirements.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Format Constraints, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.8 Transform Constraints

Identity, translation and selected scale paths have the widest acceleration; general rotation/shear can require host geometry work.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Transform Constraints, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.9 Clipping Strategy

Rectangular clipping maps efficiently to VDP state and should be preferred to application-side primitive fragmentation.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Clipping Strategy, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.10 Hardware-Only Qualification

HARDWARE_ONLY policy lets qualification detect when a critical operation would fall back to software.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Hardware-Only Qualification, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.11 Measuring Performance

Measure completed operation groups with explicit synchronization at the measurement boundary.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Measuring Performance, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.12 Memory Bandwidth

High-resolution scanout consumes framebuffer bandwidth and can reduce otherwise accelerated drawing/blit throughput.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Memory Bandwidth, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.13 Driver and Bridge Effects

Host bridge, PCI transfer size and driver command-buffer policy influence submission performance.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Driver and Bridge Effects, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

9.14 Acceleration Checklist

Record device, format, placement, state, transform, policy and completed throughput for each critical path.

Key Points

- Acceleration belongs to the complete operation/state combination, not merely the function name.
- Software fallback is permitted only when documented semantics can be preserved.
- Qualification should distinguish host submission time from completed rendering time.

Programming Notes

For Acceleration Checklist, use capability queries and statistics to confirm the backend selected during testing; never infer hardware acceleration solely from VDP model or low CPU usage.

Chapter 10 - Errors, Threading and Performance

10.1 Error Classes

Errors are grouped into validation, object/state, resource, capability, device/backend, synchronization and format categories.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Error Classes as part of normal lifecycle design rather than an exceptional afterthought.

10.2 Synchronous Errors

Invalid handle/pointer/enum/range conditions are normally returned directly by the detecting function.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Synchronous Errors as part of normal lifecycle design rather than an exceptional afterthought.

10.3 Asynchronous Errors

Device command failures can be discovered on a later Flush, Finish, readback, swap or callback boundary.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Asynchronous Errors as part of normal lifecycle design rather than an exceptional afterthought.

10.4 Device Lost Recovery

Device loss requires diagnostic capture, resource release/recreation, display-surface reacquisition and application redraw.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Device Lost Recovery as part of normal lifecycle design rather than an exceptional afterthought.

10.5 Surface Lost Recovery

Lost off-screen surfaces must be recreated from application-owned source data or regenerated caches.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Surface Lost Recovery as part of normal lifecycle design rather than an exceptional afterthought.

10.6 Timeout Handling

Timeouts must be classified by completion, vertical-blank or device communication condition before recovery.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Timeout Handling as part of normal lifecycle design rather than an exceptional afterthought.

10.7 Thread Safety

Different contexts can operate on different threads subject to driver capability; shared object destruction/modification requires serialization.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Thread Safety as part of normal lifecycle design rather than an exceptional afterthought.

10.8 Error Handlers

Callbacks should be short, non-reentrant and focused on recording diagnostic context.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Error Handlers as part of normal lifecycle design rather than an exceptional afterthought.

10.9 Logging and Trace

Diagnostic trace can record API calls, state changes, backend selection and synchronization events.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Logging and Trace as part of normal lifecycle design rather than an exceptional afterthought.

10.10 Performance Measurement

Use realistic batch sizes and report median/worst completed times rather than one fastest deferred-call duration.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Performance Measurement as part of normal lifecycle design rather than an exceptional afterthought.

10.11 Common Performance Traps

Frequent Finish, repeated surface creation, CPU locking of device memory and tiny transfers can dominate cost.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Common Performance Traps as part of normal lifecycle design rather than an exceptional afterthought.

10.12 Field Diagnostic Checklist

A useful field report includes application/runtime/driver versions, hardware revision, mode, surfaces, last error, status, statistics and a minimal call sequence.

Key Points

- Capture enough state to identify the complete failing configuration.
- Do not reset hardware before recording repeatable diagnostic data.
- Exercise recovery paths during development.

Programming Notes

A robust application treats Field Diagnostic Checklist as part of normal lifecycle design rather than an exceptional afterthought.

11.1 Graphics Library 4.0 Compatibility

Version 4.1 preserves source compatibility for most documented 4.0 calls but requires recompilation for new structures/capabilities.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Graphics Library 4.0 Compatibility is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.2 Changed Clipping Rules

4.1 standardizes rectangle edge rules and intersection of out-of-surface clip rectangles.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Changed Clipping Rules is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.3 Buffer Lifetime Changes

4.1 rejects destruction of still-bound or locked surfaces rather than silently releasing them.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Buffer Lifetime Changes is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.4 Revised Transform Validation

4.1 detects transformed coordinate overflow before unsafe device submission.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Revised Transform Validation is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.5 VDP-7 New Paths

4.1 adds VDP-7 acceleration for selected polygon, scaling, cursor and display synchronization operations.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of VDP-7 New Paths is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.6 VDP-6 Behavior

VDP-6 remains supported but can use more software fallback for features introduced with VDP-7.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of VDP-6 Behavior is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.7 Vector Compatibility Library

Vector Compatibility Library 2.8 remains the preferred adapter for older Sinneslöschen vector interfaces.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Vector Compatibility Library is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.8 Deprecated Aliases

Compatibility headers retain selected 4.0 aliases; new code should use canonical 4.1 names.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Deprecated Aliases is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.9 Migration Test Plan

Compare visible output, clipping boundaries, fonts, palette behavior, surface lifecycle, resize/mode changes, recovery and performance.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Migration Test Plan is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

11.10 Deployment Migration Checklist

Stage library, driver and application changes separately and preserve a rollback/reference configuration until field acceptance.

Key Points

- Do not combine unrelated software/hardware changes when qualification can be staged.
- Use release notes to identify edge-pixel or validation corrections.
- Preserve the previously qualified runtime for comparison.

Programming Notes

Migration of Deployment Migration Checklist is complete only when rendering, lifecycle, synchronization and performance requirements are all satisfied on representative production hardware.

Chapter 12 - Programming Examples

12.1 Basic Windowed Drawing

Open a device/context, bind the front surface and render a bordered plotting region.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.2 Strip Chart Trace

Use window/viewport mapping and Polyline for sampled engineering data.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.3 Double-Buffered Frame

Render to the back buffer and present at vertical blank.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.4 Off-Screen Layer Composition

Build a static grid in an off-screen surface and blit it into each frame.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.5 Indexed Palette Ramp

Load a 16-entry engineering color ramp during vertical blank.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.6 Technical Labels

Load a font, measure strings and right-align values inside a panel.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.7 Symbol Overlay

Load the technical symbol set and draw instrumentation symbols over a plot.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.8 Host Image Upload

Transfer a host-generated image into a surface and copy it to the display.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.9 Hardware-Only Qualification

Force hardware-only policy and verify that a critical operation remains accelerated.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

12.10 Recovery Skeleton

Reopen device resources and redraw application state after device/surface loss.

Key Points

- Check every creation/bind result in production code.
- Use explicit synchronization only when the example requires completion/display phase.
- Centralize cleanup and recovery in application infrastructure.

Programming Notes

The sample intentionally omits user-interface and application-specific resource management.

Example Skeleton

```
/* Illustrative sequence; production code checks every result. */
SGL_RESULT r = SGL_OK;
/* configure objects and state for this example */
if (r != SGL_OK) handle_error(r);
```

Appendices

Appendix A - Public Types and Constants

Type / Prefix	Purpose
SGL_DEVICE	Opaque graphics device handle
SGL_CONTEXT	Opaque graphics context handle
SGL_SURFACE	Opaque surface handle
SGL_FONT	Font resource handle
SGL_CURSOR	Cursor resource handle
SGL_POINT	Signed coordinate pair
SGL_RECT	Rectangle bounds
SGL_MATRIX	2D homogeneous transform
SGL_COLOR	Logical color value
SGL_ERR_*	Error/result constants
SGL_CAP_*	Capability selectors
SGL_FMT_*	Pixel formats

Public extensible structures begin with a size field. Initialize size to sizeof(struct) and clear unused members to zero unless a function page states otherwise.

Appendix B - Pixel Format Summary

Format	Storage	Notes
SGL_FMT_INDEX8	8 bits/pixel	Palette index; 256 entries
SGL_FMT_RGB565	16 bits/pixel	5 red, 6 green, 5 blue
SGL_FMT_XRGB8888	32 bits/pixel	8-bit RGB; high byte reserved
SGL_FMT_MONO1	1 bit/pixel	Host/resource monochrome
SGL_FMT_A1_MASK	1 bit/pixel	Cursor/glyph mask

PITCH RULE
Always use the pitch returned by the library. Width multiplied by nominal bytes-per-pixel is not guaranteed to equal row pitch.

Appendix C - Error Code Quick Reference

Code	Meaning / Initial Action
SGL_ERR_INVALID_POINTER	Correct caller pointer/structure
SGL_ERR_INVALID_HANDLE	Handle does not identify valid object
SGL_ERR_NO_CONTEXT	No current context
SGL_ERR_NO_SURFACE	Required surface not selected
SGL_ERR_OUT_OF_MEMORY	Host/device allocation failed
SGL_ERR_UNSUPPORTED_FORMAT	Format/resource unsupported
SGL_ERR_NOT_ACCELERATED	Hardware-only policy rejected fallback
SGL_ERR_SURFACE_LOST	Recreate application surface
SGL_ERR_DEVICE_LOST	Perform device recovery
SGL_ERR_TIMEOUT	Requested event did not occur
SGL_ERR_RANGE	Numeric/coordinate value out of range
SGL_ERR_OBJECT_BUSY	Object is current/bound/locked/referenced

Appendix D - Capability Selectors

Capability	Interpretation
SGL_CAP_HW_LINES	Hardware line path
SGL_CAP_HW_POLYGON	VDP polygon/triangle path
SGL_CAP_HW_SCALE_BLIT	Scaled blit path
SGL_CAP_HW_CURSOR	Hardware cursor overlay
SGL_CAP_INDEXED_DISPLAY	Indexed display mode available
SGL_CAP_MAX_SURFACE_WIDTH	Maximum surface width
SGL_CAP_MAX_SURFACE_HEIGHT	Maximum surface height
SGL_CAP_DEVICE_MEMORY	Device-memory surfaces
SGL_CAP_THREAD_CONTEXTS	Independent thread contexts
SGL_CAP_ARBITRARY_TEXT_ROTATION	Non-orthogonal text support

Re-query capabilities after major device recovery or driver replacement. A supported feature does not guarantee acceleration for every state combination.

Appendix E - Raster Operation Summary

ROP	Use
SGL_ROP_COPY	Normal drawing/copy
SGL_ROP_XOR	Transient overlays
SGL_ROP_AND	Bitwise AND where supported
SGL_ROP_OR	Bitwise OR where supported
SGL_ROP_INVERT	Destination inversion
SGL_ROP_NOP	No destination modification

Raster operations are defined on stored pixel values, not semantic colors; visual results vary with pixel format and palette.

Appendix F - Rectangle and Clipping Rules

- Transfer/fill rectangles use left/top inclusive and right/bottom exclusive bounds.
- Empty rectangles normally succeed without drawing.
- Effective destination clipping is application clip intersected with surface bounds.
- Source clipping for blits adjusts destination correspondence.
- Surface boundary protection cannot be disabled.
- General transformed geometry is clipped after transform and before raster write.

Appendix G - Function Index A-M

Function	Page
sglGetVersion	123
sglEnumerateDevices	124
sglOpenDevice	125
sglCloseDevice	126
sglGetDeviceInfo	127
sglCreateContext	128
sglDestroyContext	129
sglMakeCurrent	130
sglGetCurrentContext	131
sglFlush	132
sglFinish	133
sglResetContext	134
sglQueryCapability	135
sglGetFrontSurface	136
sglCreateSurface	137
sglDestroySurface	138
sglGetSurfaceInfo	139
sglBindSurface	140
sglSetReadSurface	141
sglLockSurface	142
sglUnlockSurface	143
sglSetDrawBuffer	144
sglSwapBuffers	145
sglSetDisplayStart	146
sglClear	147
sglHostToSurface	148
sglSurfaceToHost	149
sglCopyRect	150
sglBlit	151
sglScaleBlit	152
sglPushState	153
sglPopState	154
sglSetColor	155
sglSetBackgroundColor	156
sglSetRasterOp	157
sglSetWriteMask	158
sglSetLineStyle	159
sglSetLineWidth	160
sglSetFillPattern	161
sglSetPatternOrigin	162
sglLoadIdentity	163
sglLoadMatrix	164
sglMultiplyMatrix	165
sglTranslate	166
sglScale	167

Appendix H - Function Index N-Z

Function	Page
sglRotate	168
sglSetWindow	169
sglSetViewport	170
sglSetOrigin	171
sglSetClipRect	172
sglEnableClip	173
sglPoint	174
sglPoints	175
sglLine	176
sglPolyline	177
sglRectangle	178
sglFillRectangle	179
sglTriangle	180
sglFillTriangle	181
sglTriangles	182
sglPolygon	183
sglFillPolygon	184
sglCircle	185
sglArc	186
sglSetPaletteEntry	187
sglSetPaletteRange	188
sglGetPaletteEntry	189
sglSetColorMap	190
sglLoadFont	191
sglSelectFont	192
sglDestroyFont	193
sglSetTextColor	194
sglSetTextRotation	195
sglDrawText	196
sglMeasureText	197
sglLoadSymbolSet	198
sglDrawSymbol	199
sglCreateCursor	200
sglDestroyCursor	201
sglSetCursorImage	202
sglSetCursorPosition	203
sglShowCursor	204
sglWaitVBlank	205
sglGetStatus	206
sglGetLastError	207
sglSetErrorHandler	208
sglGetStatistics	209
sglBeginBatch	210
sglEndBatch	211
sglSetAcceleration	212

Appendix I - Glossary

Term	Meaning
acceleration path	Execution using supported VDP hardware
bound surface	Current draw destination
context	Graphics state/submission object
device surface	Surface in graphics-device memory
fallback	Software/alternate implementation
glyph cache	Reusable rasterized font/symbol storage
logical color	Application color converted for destination
pitch	Byte distance between row starts
scanout	Display fetch from framebuffer
state stack	Saved groups of graphics state
surface lost	Device storage requiring recreation
vertical blank	Display interval outside active lines

Appendix J - Support and Document Identification

Technical Support

North American Technical Support: 1-800-555-0142

Direct: 408-555-0186

Fax: 408-555-0127

E-mail: support@sinnesloschen.com

Documentation Services

Fax: 408-555-0163

E-mail: documentation@sinnesloschen.com

Software Identification

When reporting a Graphics Library issue, preserve the output of `sglGetVersion()`, the applicable display-driver version, VDP hardware model/revision and the minimum call sequence required to reproduce the behavior.

Document Identification

RM-GL41-01 Revision A - June 2, 1998. Preserve this document number and revision when citing API behavior in OEM qualification, service records or application documentation.

Sinneslöschen Systems Corporation

1450 Meridian Technology Drive

Sunnyvale, CA 94089 USA